# LAB ASSIGNMENT-7

CSN-361

Submitted by:
Ritik Kumar 17114063

**Problem Statement 1:** Transmit a binary message (from a sender to a receiver) using socket programming in C and report whether the received msg is correct or not; using the following error detection algorithms:

1. Single Parity Check
2. Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check (CRC)

## Algorithm

1. Create a sender with a socket it is listening to
2. Create a receiver to connect to the socket
3. the sender requests the user for relevant information and passes it to the receiver by sending the relevant parameters and errors using the buffer.
4. The sender encodes the data according to the 1 of the 4 techniques
5. The receiver receives, parses and checks if the data was correctly encoded and transmitted based on the algorithm

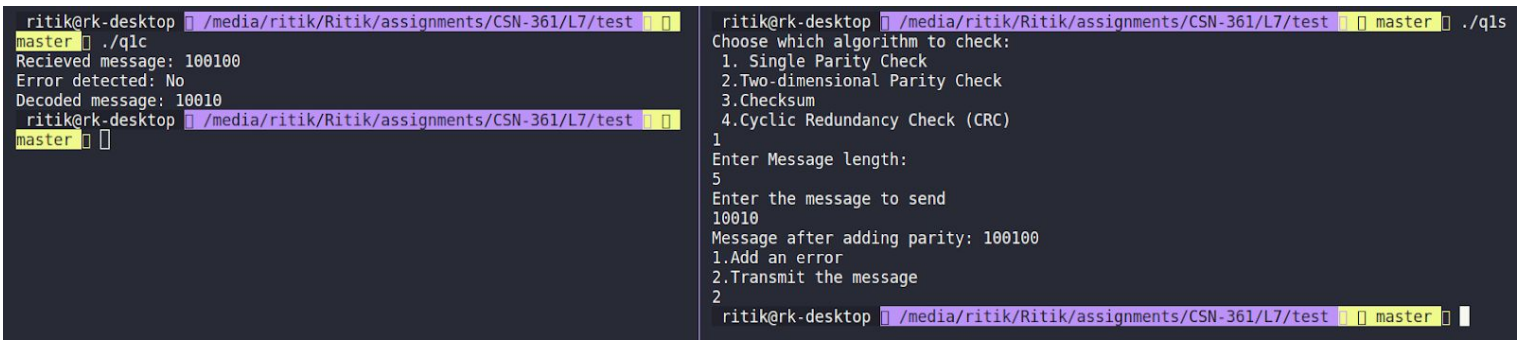
## Data Structure

Char, int arrays, flags, sockets

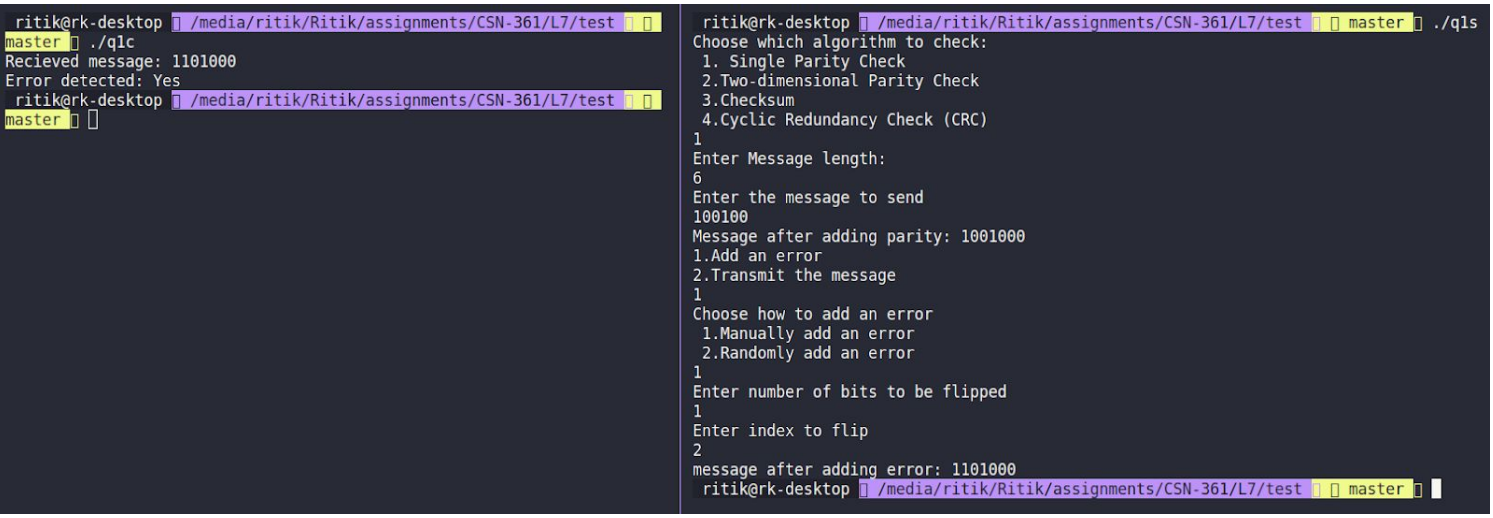
# Working ScreenShot

## 1. Single Par

(no error)




(error)




## 2. Two-dimensional Parity Check

(no error)

```
ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test  master  ./q1c
Number of segments recieved: 3
Recieved message: 11111111
Error detected: No
Decoded message: 111
ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test  master
```

```
ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test  master  ./q1s
Choose which algorithm to check:
 1. Single Parity Check
 2.Two-dimensional Parity Check
 3.Checksum
 4.Cyclic Redundancy Check (CRC)
2
Enter Message length:
4
Enter Number of segments of message
3
Enter 3 segments of 4 bits message
1011
1101
1111
Two dimensional parity matrix
1 1
1 1
1 1
1 1
1.Add an error
2.Transmit the message
2
Transmitted message: 11111111
ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test  master
```

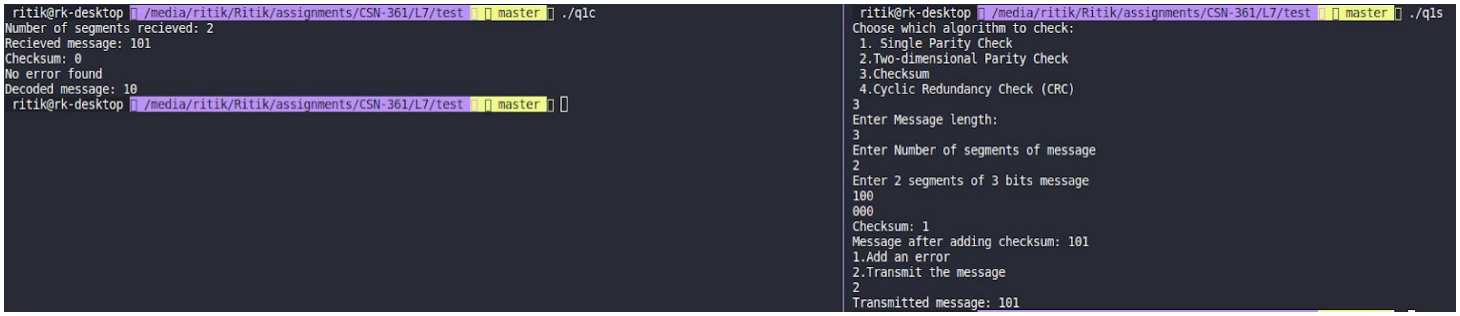
(error)

```
ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test  master  ./q1c
Number of segments recieved: 3
Recieved message: 00001011
Found Error at Row: 3
ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test  master
```

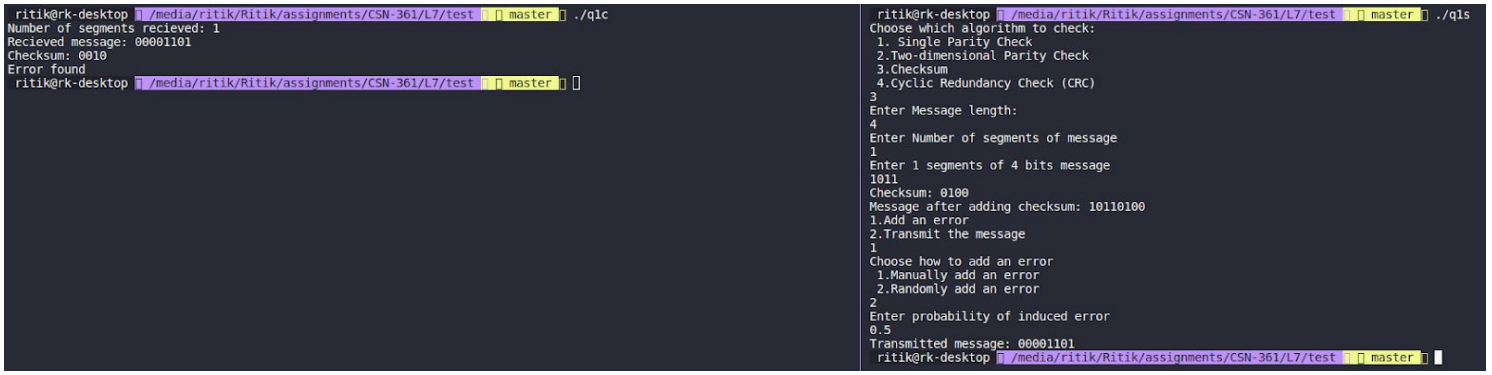
```
ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test  master  ./q1s
Choose which algorithm to check:
 1. Single Parity Check
 2.Two-dimensional Parity Check
 3.Checksum
 4.Cyclic Redundancy Check (CRC)
2
Enter Message length:
4
Enter Number of segments of message
3
Enter 3 segments of 4 bits message
1011
1101
0001
Two dimensional parity matrix
1 1
1 1
0 0
0 0
1.Add an error
2.Transmit the message
1
Choose how to add an error
 1.Manually add an error
 2.Randomly add an error
2
Enter probability of induced error
0.3
Two dimensional parity matrix after adding error
0 0
0 0
1 0
1 1
Transmitted message: 00001011
```

# 3. Checksum

(no error)




(error)




# 4. CRC

(No error)

```
itik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test    
ster  ./q1c
visor length recieved: 4
visor recieved: 1101
cieved message: 100100001
mainder: 000
 error found
coded message: 100100
itik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test    
ster  
```

```
 ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test    master  ./q1s
Choose which algorithm to check:
 1. Single Parity Check
 2.Two-dimensional Parity Check
 3.Checksum
 4.Cyclic Redundancy Check (CRC)
4
Enter length of Divisor:
4
Enter Divisor:
1101
Enter Message length:
6
Enter the message to send
100100
Remainder: 001
Message after CRC: 100100001
1.Add an error
2.Transmit the message
2
Transmitted message: 100100001
 ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test    master  
```

Error

```
ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test    
aster  ./q1c
ivisor length recieved: 4
ivisor recieved: 1101
ecieved message: 011010111
emainder: 111
rror found
ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test    
aster  
```

```
 ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test    master  ./q1s
Choose which algorithm to check:
 1. Single Parity Check
 2.Two-dimensional Parity Check
 3.Checksum
 4.Cyclic Redundancy Check (CRC)
4
Enter length of Divisor:
4
Enter Divisor:
1101
Enter Message length:
6
Enter the message to send
100100
Remainder: 001
Message after CRC: 100100001
1.Add an error
2.Transmit the message
1
Choose how to add an error
 1.Manually add an error
 2.Randomly add an error
2
Enter probability of induced error
0.3
Transmitted message: 011010111
 ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test    master  
```

---

**Problem Statement 2:** Transmit a binary message (from a sender to a receiver) using socket programming in C. Using Hamming code detect and correct errors in the transmitted message, if any.

# Algorithm

1. Create a sender with a socket it is listening to
2. Create a receiver to connect to the socket
3. the sender requests the user for relevant information on size and data and error required and passes it to the receiver by sending the relevant parameters and errors using the buffer.
4. The seder encode it using the Hamming code to generate the final message
5. The receiver receives, parses and checks if the data was correctly encoded and transmitted based on the Hamming code and finds if the code was correct and displays the correct code to the user correcting the error

# Data Structure

Char, int arrays, flags, sockets

# Code

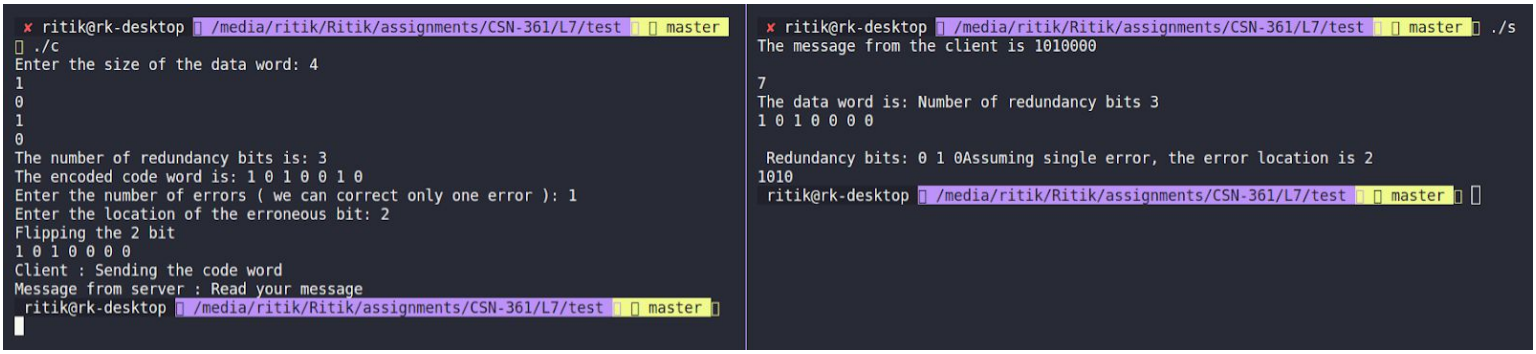
## Client

```
k=1;
for(i=1,j=0;j<dsize;i++)
{
    //If it is location of redundancy bit
    if(i==k)
    {
        df[i]=-1;
        k=k*2;
        nrb++;
    }
    //If it is location of data bit
    else
    {
        df[i]=dword[j];
        j++;
    }
    l++;
}

//Some more temp variables and arrays
int i1,i2,i3,i4,i5,i6,i7;
int a1[4],a2[4];

i5=0;       //Holds position of rb[]

//Scan Whole frame and calculate all redundancy bits
for(i1=1;i1<=l;i1++)
{
    //check if it is redundancy bit which is to be calculated
    if(df[i1]==-1)
    {
        //Get the position of 1 from binary representation
        i4 = -1;    //Hold position of 1

        //Converting redundancy bit position into binary form and get position of 1
        i7=i1;
        while (i7>0)
        {
            i4++;
            if(i7==1)
            {
                break;
            }
            else
            {
                i7=i7/2;
            }
        }

        i3=0; // Consider even parity

        //Calculating redundancy bit and parity
        for(i2=1;i2<=l;i2++)
        {
            //reset a1 frame
            for(i6=0;i6<4;i6++)
            {
                a1[i6]=-1;
            }

            //find binary conversion for each position
            i7=i2;
            i6=0;
            while(i7>0)
            {
                if(i7==1)
                {
                    a1[i6]=1;
                    break;
                }
                else if(i7==0)
                {
                    a1[i6]=0;
```

Server

```
int k;
l = csize;
i5 = 0;
k = 1;
//Scan Whole frame
for(i1=1;i1<=l;i1++)
{
    //Do processing only for redundancy bits
    if (i1==k)
    {
        //Get the position of 1 from binary representation
        i4 = -1; //Hold position of 1

        //Converting redundancy bit into binary form and get position of 1
        i7=i1;
        while (i7>0)
        {
            i4++;
            if(i7==1)
            {
                break;
            }
            else
            {
                i7=i7/2;
            }
        }

        i3=0; // Consider even parity

        //Calculating redundancy bit and parity
        for(i2=1;i2<=l;i2++)
        {
            //reset a1 frame
            for(i6=0;i6<4;i6++)
            {
                a1[i6]=-1;
            }

            //find binary conversion
            i7=i2;
            i6=0;
            while(i7>0)
            {
                if(i7==1)
                {
                    a1[i6]=1;
                    break;
                }
                else if(i7==0)
                {
                    a1[i6]=0;
                }
                else
                {
                    a1[i6]=i7%2;
                    i7=i7/2;
                }
                i6++;
            }

            //check that the binary conversion is having bit at specific position or not
            //if yes calculate parity

            if(a1[i4]==1)
            {
                if(cw[i2]==1)
                {
                    i3++;
                }
            }

        }
```

Working ScreenShot




---

## Problem Statement 3:

Write a C++ program to compress a message non-binary, can be anything like a text message or a code like hexadecimal, etc.) using the following data compression algorithm:

1. Huffman 2. Shannon-Fano

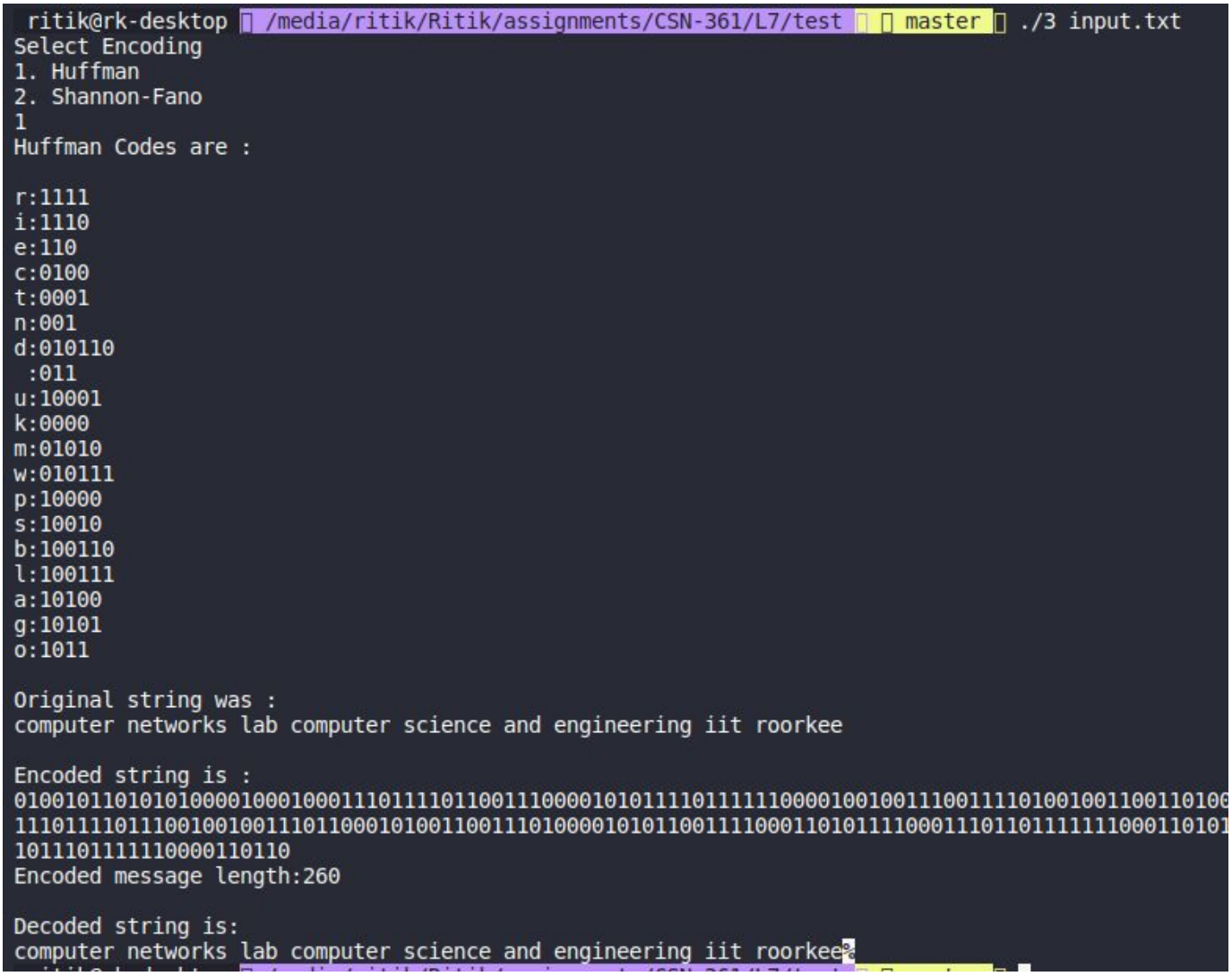
(STL use is allowed and recommended in this question)

# Algorithm

**Huffman & Shannon-Fano** are used for constructing a prefix code based on a set of symbols and their probabilities. The main objective is to generate a prefix code to encode the file data.
Pass the file to the program
Select the algorithm
Encode the files
Save the results in output.txt file

# Working ScreenShot

Huffman



Shannon-Fano

```
 ritik@rk-desktop  /media/ritik/Ritik/assignments/CSN-361/L7/test  master  ./3 input.txt
Select Encoding
1. Huffman
2. Shannon-Fano
2

19
e       0.151515        00
        0.121212        010
n       0.090909        0110
r       0.090909        0111
i       0.075758        100
o       0.075758        1010
c       0.060606        1011
t       0.060606        1100
a       0.030303        11010
g       0.030303        110110
k       0.030303        110111
m       0.030303        11100
p       0.030303        111010
s       0.030303        111011
u       0.030303        111100
b       0.015152        111101
d       0.015152        111110
l       0.015152        1111110
w       0.015152        1111111

10111010111001110101111001100000111010011000110011111111101001111101111110110101111110110101111010101011101011100111010111100110000011101011101110111100000111010111011101111000001101011000101101001101111100100001101101101000110000001111000110110110010100100110001001111010101001111101110000

Length of encoded message : 266
```