60004220215   C-186

NAME: Vruddhi Shah   STD.:   DIV.:

DATE :

PAGE :

Computer Networks
Experiment 3

Aim: To implement error detection and correction

Theory:

1. CRC - The full form of this is cyclic redundancy check. It uses a divisor polynomial that both the sender and receiver have.
The message is divided by the divisor. If the remainder is 0, then we conclude there are no errors, Otherwise we conclude error exists.
The check value is a redundancy and the algorithm is based on cyclic codes.

Eg: Divison = 1101   l = 4
Message : 10010

```
               111101
       1101 | 10010000
              1101
              ----
               1000
               1101
               ----
                1010
                1101
                ----
                 1110
                 1101
                 ----
                  011   → there is an error
```

2. Hamming code:
Hamming codes are a family of linear error correcting codes. If the message length is $2^r - r - 1$, then

that needs r parity bits, getting the total number of bits up to $2^r - 1$. The parity check matrix of a Hamming code is costructed by listing all colums of length r that are non-zero.

Eg: Hammsing (7, 4)

All powers of 2 are parity bits

$$\begin{array}{ccccccc} p_1 & p_2 & d_1 & p_3 & d_2 & d_3 & d_4 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}$$

$p_1 = p_1 \oplus d_1 \oplus d_2 \oplus d_4$

$p_2 = p_2 \oplus d_1 \oplus d_3 \oplus d_4$

$p_3 = p_3 \oplus d_2 \oplus d_3 \oplus d_4$

Conclusion: Thus we implemented error detection and correction codes.

Department of Computer Engineering
Class: S.Y. B.Tech. **Semester: V**
Course Name: Computer Networks Lab

Name: Vruddhi Shah	**SAP ID: 60004220215**
Date of Performance: 21.08.24	**Date of Submission: 21.08.24**

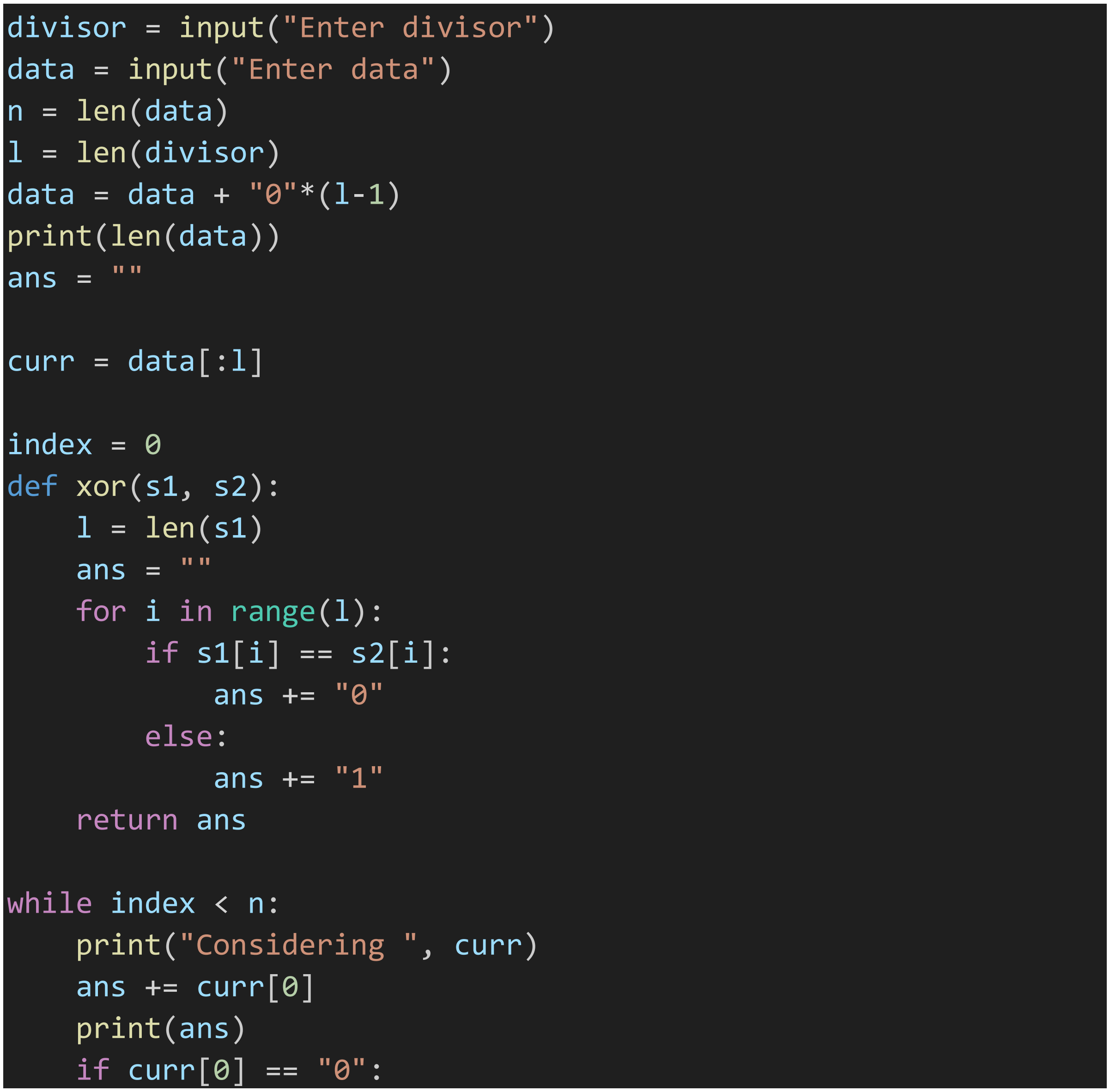
Experiment No: 3

Aim: Write a program to implement error detection and correction.

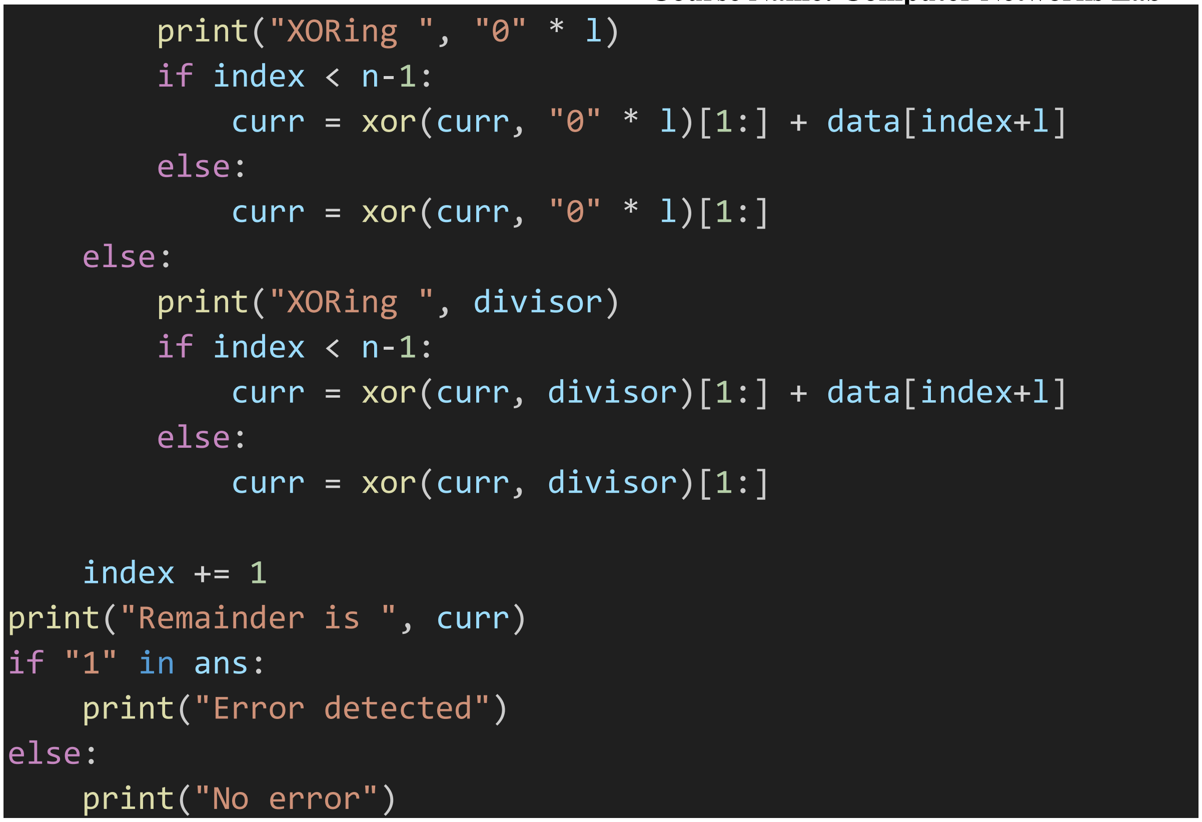
Techniques: CRC, Hamming

Program:

CRC:

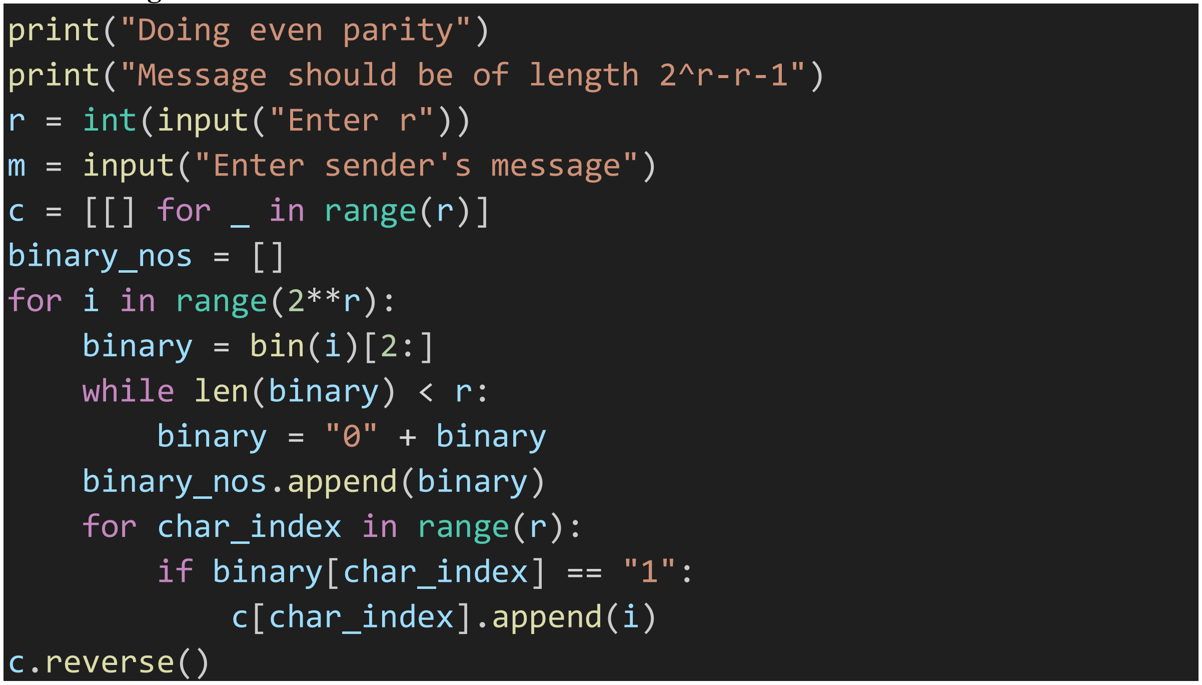


```
divisor = input("Enter divisor")
data = input("Enter data")
n = len(data)
l = len(divisor)
data = data + "0"*(l-1)
print(len(data))
ans = ""

curr = data[:l]

index = 0
def xor(s1, s2):
    l = len(s1)
    ans = ""
    for i in range(l):
        if s1[i] == s2[i]:
            ans += "0"
        else:
            ans += "1"
    return ans

while index < n:
    print("Considering ", curr)
    ans += curr[0]
    print(ans)
    if curr[0] == "0":
```

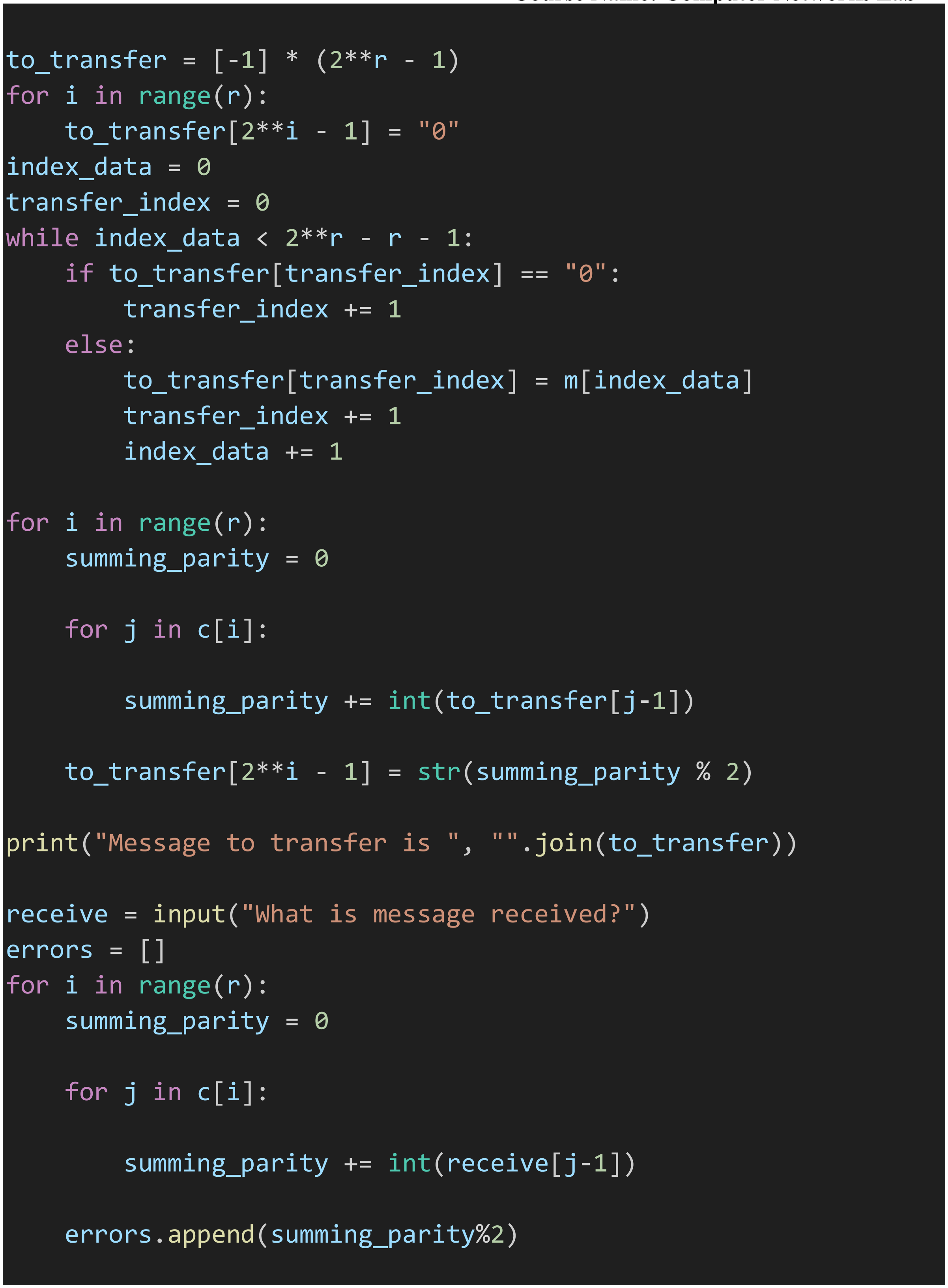


```
        print("XORing ", "0" * l)
        if index < n-1:
            curr = xor(curr, "0" * l)[1:] + data[index+l]
        else:
            curr = xor(curr, "0" * l)[1:]
    else:
        print("XORing ", divisor)
        if index < n-1:
            curr = xor(curr, divisor)[1:] + data[index+l]
        else:
            curr = xor(curr, divisor)[1:]

    index += 1
print("Remainder is ", curr)
if "1" in ans:
    print("Error detected")
else:
    print("No error")
```

Hamming:



```
print("Doing even parity")
print("Message should be of length 2^r-r-1")
r = int(input("Enter r"))
m = input("Enter sender's message")
c = [[] for _ in range(r)]
binary_nos = []
for i in range(2**r):
    binary = bin(i)[2:]
    while len(binary) < r:
        binary = "0" + binary
    binary_nos.append(binary)
    for char_index in range(r):
        if binary[char_index] == "1":
            c[char_index].append(i)
c.reverse()
```

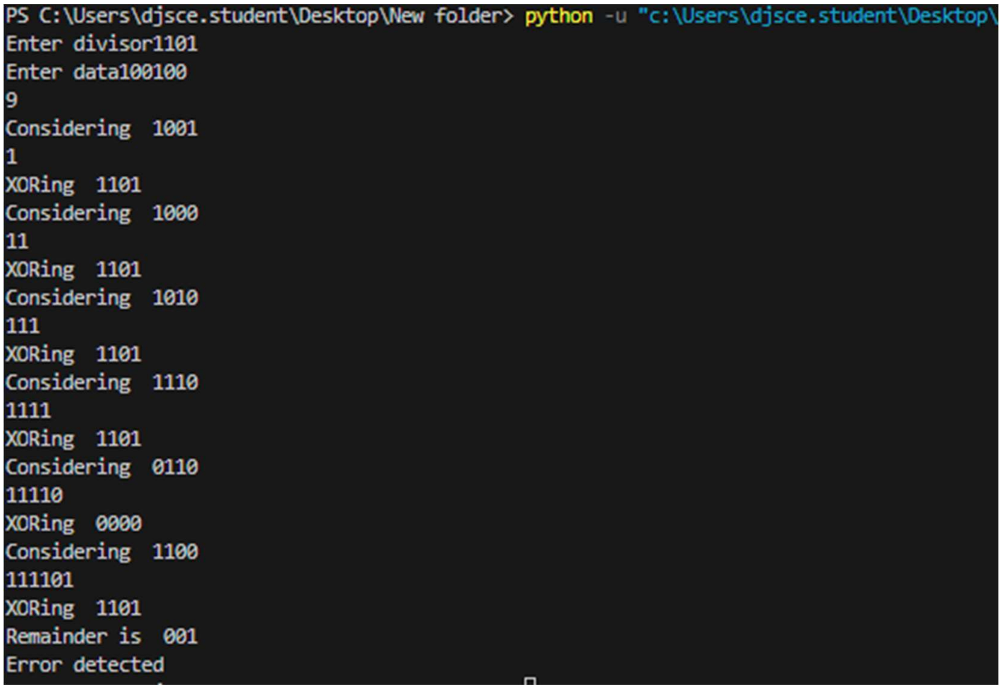


```
to_transfer = [-1] * (2**r - 1)
for i in range(r):
    to_transfer[2**i - 1] = "0"
index_data = 0
transfer_index = 0
while index_data < 2**r - r - 1:
    if to_transfer[transfer_index] == "0":
        transfer_index += 1
    else:
        to_transfer[transfer_index] = m[index_data]
        transfer_index += 1
        index_data += 1

for i in range(r):
    summing_parity = 0

    for j in c[i]:

        summing_parity += int(to_transfer[j-1])

    to_transfer[2**i - 1] = str(summing_parity % 2)

print("Message to transfer is ", "".join(to_transfer))

receive = input("What is message received?")
errors = []
for i in range(r):
    summing_parity = 0

    for j in c[i]:

        summing_parity += int(receive[j-1])

    errors.append(summing_parity%2)
```

```
errors = reversed(errors)
errors = [str(e) for e in errors]
def binaryToDecimal(binary):

    decimal, i = 0, 0
    while(binary != 0):
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1
    return decimal
binary_errors = "".join(errors)


error_pos = binaryToDecimal(int(binary_errors))
print("Error at", error_pos)
```

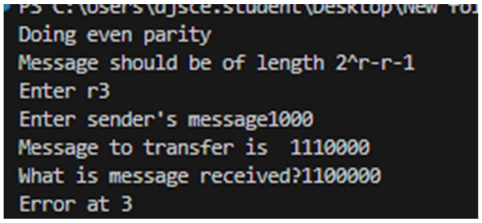
Screenshots:

CRC:

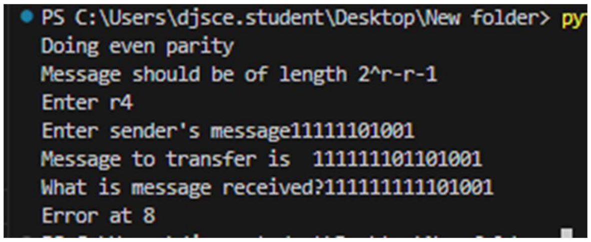


```
PS C:\Users\djsce.student\Desktop\New folder> python -u "c:\Users\djsce.student\Desktop\
Enter divisor1101
Enter data100100
9
Considering  1001
1
XORing  1101
Considering  1000
11
XORing  1101
Considering  1010
111
XORing  1101
Considering  1110
1111
XORing  1101
Considering  0110
11110
XORing  0000
Considering  1100
111101
XORing  1101
Remainder is  001
Error detected
```

Hamming:



```
PS C:\Users\djsce.student\Desktop\New fo
Doing even parity
Message should be of length 2^r-r-1
Enter r3
Enter sender's message1000
Message to transfer is  1110000
What is message received?1100000
Error at 3
```



```
PS C:\Users\djsce.student\Desktop\New folder> py
 Doing even parity
 Message should be of length 2^r-r-1
 Enter r4
 Enter sender's message11111101001
 Message to transfer is  111111101101001
 What is message received?111111111101001
 Error at 8
```

Conclusion:

Thus, we have studied and implemented error detection and correction.