# Practical - 6

Aim :

write a program to implement error detection and correction using Hamming code concept. Make a test run to input data stream and verify error correction.

Error correction at Data Link layer :

Hamming code is set of error correction codes that can be used it detect and correct the errors that can when the data is transmitted from the sender to the reciever. It is a technique developed by R. W. Hamming for error correction.

create sender program with below features:

1) Input to sender file should be a text any length program should convert the text to binary.

2) Apply hamming code concept on the binary data add redundant bits to it.

create a reciever program with below features:

1) Receiver program should read the input from channel file.

2) Apply hamming code on the binary data to check for errors.

3) If there is an error, display the position of error.

4) Then remove the redundant bits and convert the binary data to ascii and display the output.

Program:

```
def calc_r(l):
    for i in range(l):
        if(2**i >= l+i+l):
            return i
def pos_red_bits(b,r)
    j,k = 0,1
    bits = " "
    for i in range(1, len(b)+r+1):
        if(i == 2**j):
            bits +="0"
            j+=1
        else:
            bits += b[-1*k]
            k+=1
    return bits[::-1]
def calc_parity(arr,r):
    n = len(arr)
    for i in range(r):
        val = 0
        for j in range(1, n+1)
            if (j & (2**i)) == (2**i)):
                val = val ^ int(arr[-1*j])
        res = res + val * (10**i)
    return int(str(res),2)
def flip(data,pos)
    if pos<1 or pos > len(data):
        print("Invalid position!")
        return data.
    data_list = list(data)
    data_list[pos-1]='1' if data_list[pos-1]
                    =='0' else '0'

    return ''.join(data_list)
def bin_to_dec(b):
    return int(b,2)
s = input("Enter a string to encode:")
bin_val = ''.join([bin(ord(c))[2:].zfill(8)
                  for c in s])
print(f"Binary representation of {s}:{bin
                  _val}")
l = len(bin_val)
```

```
r = calc_r(l)
print(f"Number of redundant bits: {r}")
pos = pos_red_bits(bin_val, r)
enc_data = calc_party(pos, r)
print(f"Data with redundant bits:
                        {enc_data}")
while True:
    err_pos = int(input(f"Enter the
            position of the bit to flip
        (1-based index 1 to {len(enc_
                   data)}): "))
    if err_pos in [2**i for i in range(r)]:
        print("cannot flip a redundant bit
        position. Please enter a valid position")
continue
else:
    err_pos_detected = err_detected
    err_pos_left = len(enc_data_err) -
                  err_pos_detected + 1
    bin_err_pos = bin(err_pos_left)[2:]
              zfill(4)
    dec_err_pos = bin_to_dec(bin_err_pos
    print(f"Error detected at position: {err_pos
                                   _left}")
```

```
if correct = 'yes':
    corrected_data = flip(enc_data_err,
                        err_pos_left)
    print(f"corrected data: {corrected_data}")
else:
    print("Error was not corrected")
```

Output:

```
Enter a string to encode: Hi
Binary representation of 'Hi': 0100100001
                                  101001
Number of redundant bits: 5
Data with redundant bits: 0100100000110
                          000100

Enter the position of the bit to flip (1 to 2
                       : 6

Data with error introduced: 0100110000
                        11001000100

Error detected at position: 6
Binary error position: 0110, decimal: 6
Corrected data: 010010000011001000010
=== code execution successful :::
```

Result:

Thus the error detection and correction using Hamming code concept was successfully implement and executed.