Exp.No: Practical - 06
DATE: 20/8/24

AIM:
To 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 feature.

Error correction at Data link Layer:

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

Sender program features:

1) Input to sender file should be a text of any length. Program should convert it to binary.

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

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

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

Sender program:

```
def detect_error(hamming_code):
    n = len(hamming_code
```

Code:

```
import math
def char_to_binary(ch):
    binary = []
    for i in range(7,-1,-1):
        binary.append((ord(ch)>>i)&1)
    return binary

def calculate_parity_bits(hamming_code,n,r):
    for i in range(r):
        parity_pos = 2**i
        parity = 0
        for j in range(parity_pos-1,n+1,2*
                       parity_pos):
            for k in range(j, j+parity_pos):
                if k <= n:
                    parity = hamming_code[k]
        hamming_code[parity_pos] = parity

def generate_hamming_code(data_bits,m):
    r=0
    n=m
    while n+r+1>2**r:
        r+=1
    n=m+r
    hamming_code = [0]*(n+1)
```

```
j=0
k=0
for i in range(1,n+1):
    if i==2**k:
        k+=1
    else:
        hamming_code[i]=data_bits[j]
        j+=1

calculate_parity_bits(hamming_code,n,r)
return hamming_code,n,r

def detect_and_correct_error(hamming_code,
    error_pos=0
    for i in range(r):
        parity_pos=2**i
        parity=0
        for j in range(parity_pos,n+1,2*parity_pos):
            for k in range(j,j+parity_pos):
                if k<=n:
                    parity=hamming_code[k]
        if parity!=0:
            error_pos+=parity_pos
    return error_pos
```

```
def binary_to_char(binary):
    output = 0
    for i in range(0, len(binary), 8):
        ch = 0
        for j in range(8):
            ch1 = (binary[i+j] << (7-j))
        output += chr(ch)
    return output

def main():
    input_string = input("Enter the input string")
    binary = []
    for ch in input_string:
        binary.extend(char_to_binary(ch))
    data_bits = binary[i]
    hamming_code, n, r = generate_hamming_code
                         (data_bits, len(data_bits))
    print("Generated hamming code:", ''.join
                         (map(str, hamming_code[:])))
    error_pos = -1
    while True:
        error_pos = int(input("Enter the position"))
```

```
if error_pos > 0 and any(error_pos == 2**k
    for k in range(r)):
    print("Error cannot be introduced in a
        redundant (parity) position, please choose
        another position.")

    dif error_pos < 0 or error_pos > n:
        print("Invalid position. Please enter a
            position between 1 and", n)

    else:
    break

if error_pos > 0:
    hamming_code[error_pos] ^= 1
    print("Hamming code with error:",
        ''.join(map(str, hamming_code[1:])))

detected_error_pos = detect_and_cor
                    _error(hamming_code,
                        n, r)

if detected_error_pos == 0:
    print("No error detected")

else
    print(f"Error detected at position:
        {detected_error_pos}")
```

```
binary_error_pos = format(detected_error_pos,
                          'b')

print(f"corrected bit at position
    {detected_error_pos}(binary: {binary_
                                  error_pos}):
        {hamming_code[detected_error_pos]}")

hamming_code[detected_error_pos]
print("corrected hamming code:''.join(map(
        str, hamming_code[1:5]))

corrected_data_bits=[]

k=0
for i in range(1,n+1):
    if i&(i-1)==2**k
        corrected_data_bits.append
                        (hamming_code[i]

    else:
        k+=1
corrected_string=binary_to_char(
                        corrected_data_bits)
print("corrected string:", corrected_string):

if __name__=="__main__":
    main()
```

OUTPUT:

Enter input string : apple

Generated hamming code: 110011000
001111000001111 000001111 011000 110101

Enter position to simulate error : 3

Hamming code with error: 111011 000001011111
000001110000111 000001110 000111011000110101

Error detected at position: 3
corrected bit at position 3 (binary : 1)

Corrected hamming code : 110011000001011111000
1110000011101100011001 01

Corrected string : apple

Result:

Thus, the program is executed successfully.