

## **EXPERIMENT – 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 feature.

i	power of 2	action	res'	y'	%
1	1=2**0=yes	insert 0'	0'	1	1
2	2=2**1=yes	insert 0'	00'	2	1
3	3=2**2(NO)	insert data	001'	2	2
4	4=2**2(YES)	insert 0	0010'	3	2
5	5=2**3	insert data	00100'	3	3
6	6=2**3	insert data	001001'	3	4
7	7=2**3	insert data	0010011'	3	5
8	8=2**3(YES)	insert 0'	00100110'	4	5
9	9=2**3(no)				

**CODE: -**

```
def calcRedundantBits(m):
    # Use the formula  $2^r \geq m + r + 1$ 
    for i in range(m):
        if(2**i >= m + i + 1):
            return i

def posRedundantBits(data, r):
    # Redundancy bits are placed at the positions
    j = 0
    k = 1
    m = len(data)
    res = ''
    # If position is power of 2 then insert '0' Else append the data
    for i in range(1, m+r+1):
        if(i == 2**j):
            res = res + '0'
            j += 1
        else:
            res = res + data[-1 * k]
            k += 1
    # The result is reversed since positions are counted backwards. (m + r+1
    ... 1)
    return res[::-1]

def calcParityBits(arr, r):
    n = len(arr)
    # For finding rth parity bit, iterate over
```

```

# 0 to r - 1
for i in range(r):
    val = 0
    for j in range(1, n + 1):

        # If position has 1 in ith significant
        # position then Bitwise OR the array value
        # to find parity bit value.
        if(j & (2**i) == (2**i)):
            val = val ^ int(arr[-1 * j])
            # -1 * j is given since array is reversed

    # String Concatenation
    # (0 to n - 2^r) + parity bit + (n - 2^r + 1 to n)
    arr = arr[:n-(2**i)] + str(val) + arr[n-(2**i)+1:]
return arr

def detectError(arr, nr):
    n = len(arr)
    res = 0

    # Calculate parity bits again
    for i in range(nr):
        val = 0
        for j in range(1, n + 1):
            if(j & (2**i) == (2**i)):
                val = val ^ int(arr[-1 * j])

        # Create a binary no by appending
        # parity bits together.

        res = res + val*(10**i)

    # Convert binary to decimal
    return int(str(res), 2)

# Enter the data to be transmitted
data = '1011001'

# Calculate the no of Redundant Bits Required
m = len(data)
r = calcRedundantBits(m)

# Determine the positions of Redundant Bits
arr = posRedundantBits(data, r)

# Determine the parity bits
arr = calcParityBits(arr, r)

```

```

# Data to be transferred
print("Data transferred is " + arr)

# Stimulate error in transmission by changing
# a bit value.
# 10101001110 -> 11101001110, error in 10th position.

arr = '10101001110'
print("Error Data is " + arr)
correction = detectError(arr, r)
if(correction==0):
    print("There is no error in the received message.")
else:
    print("The position of error is ",len(arr)-correction+1,"from the left")

```

## OUTPUT: -

```

1 def calculateParityBits(n):
2     # Apply the formula 2^k = n-1 to find k
3     for i in range(1, n):
4         if((2**i) >= n-1):
5             return i
6
7 def positionOfError(data, r):
8     # Generating parity bits and adding to the payload
9     k = 0
10    n = len(data)
11    res = ""
12    # If position of error is 0 then correct it, else append the data
13    for i in range(1, n+1):
14        if(i <= 2**k):
15            res += "1"
16        else:
17            res += data[i-1]
18        k += 1
19    # The result is correct since payload size added between (n+1) to (n+k)
20    return res[1:]
21
22 def calculateParityBits(n):
23     n = len(arr)
24     # Applying the formula 2^k = n-1 to find k
25     for i in range(1, n):
26         if((2**i) >= n-1):
27             return i
28
29 def detectError(data, r):
30     # Generating parity bits and adding to the payload
31     k = 0
32     n = len(data)
33     res = ""
34     # If position of error is 0 then correct it, else append the data
35     for i in range(1, n+1):
36         if(i <= 2**k):
37             res += "1"
38         else:
39             res += data[i-1]
40        k += 1
41    # The result is correct since payload size added between (n+1) to (n+k)
42    return res[1:]
43
44 def correctError(data, r):
45     # Generating parity bits and adding to the payload
46     k = 0
47     n = len(data)
48     res = ""
49     # If position of error is 0 then correct it, else append the data
50     for i in range(1, n+1):
51         if(i <= 2**k):
52             res += "1"
53         else:
54             res += data[i-1]
55        k += 1
56    # The result is correct since payload size added between (n+1) to (n+k)
57    return res[1:]
58
59 # Data to be transferred
60 print("Data transferred is " + arr)
61
62 # Stimulate error in transmission by changing
63 # a bit value.
64 # 10101001110 -> 11101001110, error in 10th position.
65
66 arr = '10101001110'
67 print("Error Data is " + arr)
68 correction = detectError(arr, r)
69 if(correction==0):
70     print("There is no error in the received message.")
71 else:
72     print("The position of error is ",len(arr)-correction+1,"from the left")

```

## RESULT: -

The code for HAMMING CODE have been executed successfully and the output is verified.