

## 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.

[illegible]

**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: -

The screenshot shows a code editor with a file named 'main.py'. The code defines three functions: `calcRedundantBits(m)`, `posRedundantBits(data, r)`, and `calcParityBits(arr, r)`. The `calcRedundantBits` function calculates the number of redundant bits based on the data length `m` and the number of parity bits `r`. The `posRedundantBits` function inserts '0' at positions that are powers of 2. The `calcParityBits` function calculates the parity for each data bit. The main code block initializes `arr = '10101001110'`, prints the error data, and calls `detectError(arr, r)`. The output window shows the following text:

```

Data transferred is 10101001110
Error Data is 1010111101
The position of error is 4 from the left

=== Code Execution Successful ===

```

## RESULT: -

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