# CN LAB 4

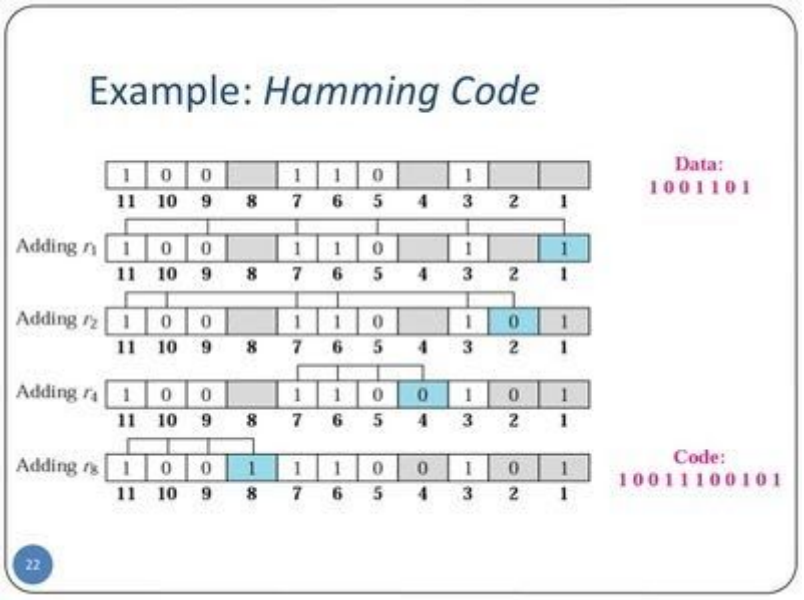
## SAHIL BONDRE: U18CO021

## Problem:

**Implement a program to detect and correct error using hamming code,**

## Solution:

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 moved or stored from the sender to the receiver. It is a technique developed by R.W. Hamming for error correction.



## Python Implementation:

```
from typing import List


def redundantBits(m: int) -> int:
    """Get number of redundant bits"""

    # 2 ^ r >= m + r + 1
    for i in range(m):
        if(2**i >= m + i + 1):
            return i


def redundantPositions(data: str, r: int) -> str:
    """Get positions of redundant bits"""
    j = 0
    k = 1
    m = len(data)
    res = ""

    for i in range(1, m + r + 1):
        if(i == 2 ** j):
            # Power of two
            res = res + "0"
            j += 1
        else:
            # not power of two
            # reversed results
            res = res + data[-1 * k]
            k += 1

    return res[::-1]


def parityBits(arr: List[int], r: int) -> str:
    """Get encoded data"""
    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

        arr = arr[:n - (2**i)] + str(val) + arr[n - (2**i) + 1:]
    return arr


def detectError(arr: List[int], nr: int) -> int:
    """Return error bit position"""
    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])


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

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


data = "1011001"
print(f"Data: {data}")
m = len(data)
r = redundantBits(m)
print(f"Number of redundant bits: {r}")

arr = redundantPositions(data, r)
print(f"Redundant bit positions: {arr}")
# Determine the parity bits
arr = parityBits(arr, r)

# Data to be transferred
print(f"Data transferred: {arr}")

# 10101001110 -> 11101001110, error in 10th position.

arr = "11101001110"
```

```
print(f"Error Data: {arr}")
correction = detectError(arr, r)
print(f"Error Position: {correction}")
```