

Experiment 2

Aim: Implementation of hamming code for error detection and correction.

Theory:

- Hamming code is a set of error-detection 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.

- Redundant Bits

⇒ Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer.

- The number of redundant bits can be calculated using the following formula:

$$2^r \geq m + r + 1$$

Where, r = redundant bits,
 m = data bits

- Even Parity bit: In the case of even parity, for a given set of bits, the number of 1's are counted.
- If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1's even number.

- IF the total number of 1's in a given set of bits is already even, the parity bit value is 0.
- Odd Parity bit : In the case of odd parity, for a given sets of bits, the number of 1's are counted.
- IF that count is even, the parity bit value is set to 1, making the total count of occurrences of 1's odd number.
- IF the total number of 1's in a given set of bits is already odd, the parity bit's value is 0.

• Algorithm :

- 1) Write the bit position starting from 1 in binary form (1, 10, 11, 100, etc).
 - 2) All the bits positions that are a power of 2 are marked as parity bits.
 - 3) All the other bit positions are marked as data bits.
 - 4) Each data bit is included in a unique set of parity bits as determined its bit position in binary form.
- Parity bit 1 covers all the bits position in binary representation includes a 1 in the least significant position (1, 3, 5, 7, etc)
 - Parity bit 2 covers all the bits positions whose binary representation include a 1

in the second position from the least significant bit (2, 3, 6, 7, etc).

- Parity bit 4 covers all the bits position whose binary representation included a 1 in the third position from the least significant bit (4-7, 12-15, etc).

- In general, each parity bits covers all bits where the bitwise AND of the parity position and the bit position is non-zero.

5) Since we check for even parity set a parity bit to 1 if the total number of ones in the position it checks is odd.

6) Set a parity bit to 0 if the total number of ones in the position it checks is even.

- For e.g.,
data bit = 1011

D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
1	0	1	P ₄	1	P ₂	P ₁

- For P₁, sections to be considered are 1, 3, 5, 7.

Here, we have to set P₁ = 1 as 3, 5, 7 = 111 in order to have even parity.

• Decide P_2

\Rightarrow For P_2 , sections to be considered 2, 3, 6, 7
Here we have to set $P_2 = 0$ as 3, 6, 7
 $= 101$ in order to have even parity.

• Decide P_4

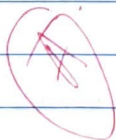
\Rightarrow For P_4 , sections to be considered are
4, 5, 6, 7.

Here we have to set $P_4 = 0$ as
5, 6, 7 = 101 in order to have the even
parity.

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	0	1	0	1

Thus, the code word which is transmitted
to the receiver = 1010101.

~~Q210101~~



Code:

```
#include <iostream>
using namespace std;

int main() {
    int data[10];
    int dataatrec[10], c, c1, c2, c3, i;

    cout << "Enter 4 bits of data one by one\n";
    cin >> data[0];
    cin >> data[1];
    cin >> data[2];
    cin >> data[4];

    data[6] = data[0] ^ data[2] ^ data[4];
    data[5] = data[0] ^ data[1] ^ data[4];
    data[3] = data[0] ^ data[1] ^ data[2];

    cout << "\nEncoded data is\n";
    for (i = 0; i < 7; i++)
        cout << data[i];

    cout << "\n\nEnter received data bits one by one\n";
    for (i = 0; i < 7; i++)
        cin >> dataatrec[i];

    c1 = dataatrec[6] ^ dataatrec[4] ^ dataatrec[2] ^ dataatrec[0];
    c2 = dataatrec[5] ^ dataatrec[4] ^ dataatrec[1] ^ dataatrec[0];
    c3 = dataatrec[3] ^ dataatrec[2] ^ dataatrec[1] ^ dataatrec[0];
    c = c3 * 4 + c2 * 2 + c1;

    if (c == 0) {
        cout << "\nNo error while transmission of data\n";
    } else {
        cout << "\nError on position " << c;
        cout << "\nData sent : ";
        for (i = 0; i < 7; i++)
            cout << data[i];

        cout << "\nData received : ";
        for (i = 0; i < 7; i++)
            cout << dataatrec[i];
    }
}
```

```

cout << "\nCorrect message is\n";

if (dataatrec[7 - c] == 0)
    dataatrec[7 - c] = 1;
else
    dataatrec[7 - c] = 0;
for (i = 0; i < 7; i++) {
    cout << dataatrec[i];
}
}
return 0;
}

```

Output:

```

>_ Console v x Shell x + ...
> sh -c make -s
> ./main
Enter 4 bits of data one by one
1
1
0
1

Encoded data is
1100110

Enter received data bits one by one
1
1
1
1
1
1
1
0

Error on position 1
Data sent : 1100110
Data received : 1111110
Correct message is
1111111> 

```