Hamming code-

1. It was developed by Richard Hamming in the 1950s
2. It is error correcting code.
3. Simple and effective code that help to improve readability of code.

Terminologies-

1. Redundant bit-
    a. Extra bit that is added in the information bit.
    b. It is added so that if error occurs in the data then we can find the error.
    c. Number of redundan bits are calculated by the formula

        2^r >= m + r + 1 where

        m = Number of input bit in data, r = number of redundant bit.
2. Parity bit-
    a. The extra redundant bit that is added.
    b. It has 2 types-
        i. Even paritybit-
            1. In the data bit number on one present in data should be in even number
        ii. Odd parity bit-
            1. In the data bit number on one present in data should be in odd number.

Working of Hamming code-

Sender Side:-

1. Find number of redundant bits by formula.
2. Assign the given data bits to the position.
3. Check the parity of bits and give values to redundant bits.
4. Send the encoded data.

Receiver Side:-

1. Calculate Parity of the received data.
2. If even parity is there:
    a. No error is caused in transfer of data.
3. If odd parity is there:
    a. Error has occurred.
    b. Then calculate parity for every redundant bit.
    c. This will give you the bit that has error.
    d. Then correct that bit and you will get the correct code word.
    e. Remove all the redundant bits and you will get the actual data that was send be sender.

Input Code-

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

int main() {
    int data[4];
    int hammingCode[7];

    cout << "Enter a 4-bit bitstream (one bit at a time):\n";
    for (int i = 0; i < 4; i++) {
        cin >> data[i];
        if (data[i] != 0 && data[i] != 1) {
            cout << "Invalid input. Please enter only 0 or 1.\n";
            return 1;
        }
    }
    // Assign data bits to Hamming code positions
    hammingCode[6] = data[0];
    hammingCode[5] = data[1];
    hammingCode[4] = data[2];
    hammingCode[2] = data[3];

    // Calculate redundant bits
    hammingCode[0] = hammingCode[2] ^ hammingCode[4] ^ hammingCode[6]; //
R1
    hammingCode[1] = hammingCode[2] ^ hammingCode[5] ^ hammingCode[6]; //
R2
    hammingCode[3] = hammingCode[4] ^ hammingCode[5] ^ hammingCode[6]; //
R4

    cout << "Calculated redundant bits:\n";
    cout << "R1: " << hammingCode[0] << endl;
    cout << "R2: " << hammingCode[1] << endl;
    cout << "R4: " << hammingCode[3] << endl;

    cout << "Hamming code (D7, D6, D5, R4, D3, R2, R1):\n";
    for (int i = 6; i >= 0; i--) {
        cout << hammingCode[i] << " ";
    }
    cout << endl;
```

```
    // Receiver Side
    int receivedCode[7];
    cout << "\nEnter the received 7-bit Hamming code:\n";
    for (int i = 6; i >= 0; i--) {
        cin >> receivedCode[i];
        if (receivedCode[i] != 0 && receivedCode[i] != 1) {
            cout << "Invalid input. Please enter only 0 or 1.\n";
            return 1;
        }
    }

    int r1 = receivedCode[0] ^ receivedCode[2] ^ receivedCode[4] ^
receivedCode[6];
    int r2 = receivedCode[1] ^ receivedCode[2] ^ receivedCode[5] ^
receivedCode[6];
    int r4 = receivedCode[3] ^ receivedCode[4] ^ receivedCode[5] ^
receivedCode[6];

    int errorIndex = r4 * 4 + r2 * 2 + r1 * 1;

    if (errorIndex == 0) {
        cout << "No error detected.\n";
    } else {
        cout << "Error detected at position: " << errorIndex << endl;
        receivedCode[errorIndex - 1] ^= 1;
        cout << "Corrected Hamming code: ";
        for (int i = 6; i >= 0; i--) {
            cout << receivedCode[i] << " ";
        }
        cout << endl;
    }

    // Extract original 4-bit data from corrected Hamming code
    cout << "Original 4-bit data: ";
    cout << receivedCode[6] << " " << receivedCode[5] << " " <<
receivedCode[4] << " " << receivedCode[2] << endl;

    return 0;
}
```

Output Code-

```
Enter a 4-bit bitstream (one bit at a time):
1 1 0 1
Calculated redundant bits:
R1: 0
R2: 1
R4: 0
Hamming code (D7, D6, D5, R4, D3, R2, R1):
1 1 0 0 1 1 0

Enter the received 7-bit Hamming code:
1 0 0 0 1 1 0
Error detected at position: 6
Corrected Hamming code: 1 1 0 0 1 1 0
Original 4-bit data: 1 1 0 1
```