HAMMING CODE

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

// Function to calculate the number of parity bits needed
int calculateParityBits(int dataBits) {
    int parityBits = 0;
    while (pow(2, parityBits) < dataBits + parityBits + 1) {
        parityBits++;
    }
    return parityBits;
}

// Function to encode the data using Hamming code with data bits entered from back
vector<int> encodeData(vector<int> data) {
    int dataBits = data.size();
    int parityBits = calculateParityBits(dataBits);
    int totalBits = dataBits + parityBits;
    vector<int> encoded(totalBits, 0);

    // Fill data bits from the end (rightmost)
    int dataIndex = dataBits - 1;
    for (int i = totalBits - 1; i >= 0; i--) {
        // Check if the position is not a power of 2 (not a parity bit)
        bool isPowerOfTwo = false;
        for (int j = 0; j < parityBits; j++) {
            if (i + 1 == pow(2, j)) {
                isPowerOfTwo = true;
                break;
            }
        }
        if (!isPowerOfTwo && dataIndex >= 0) {
            encoded[i] = data[dataIndex--];
        }
    }

    // Calculate and set the parity bits
    for (int i = 0; i < parityBits; i++) {
        int parityBitPos = pow(2, i) - 1; // Convert to 0-based index
        int sum = 0;

        // Check all bits that have (parityBitPos + 1) in their coverage
        for (int j = parityBitPos; j < totalBits; j++) {
            if (((j + 1) & (parityBitPos + 1)) != 0) {
                sum += encoded[j];
            }
        }
        encoded[parityBitPos] = sum % 2;
    }

    return encoded;
}

// Function to check for errors in the encoded data
int checkForErrors(vector<int> encoded) {
    int parityBits = calculateParityBits(encoded.size() - calculateParityBits(encoded.size()));
    int errorPosition = 0;

    for (int i = 0; i < parityBits; i++) {
        int parityBitPos = pow(2, i) - 1;
        int sum = 0;

        for (int j = parityBitPos; j < encoded.size(); j++) {
            if (((j + 1) & (parityBitPos + 1)) != 0) {
                sum += encoded[j];
            }
        }
        errorPosition += (sum % 2) * (parityBitPos + 1);
    }

    return errorPosition;
}

int main() {
    int dataBits;
    cout << "Enter the number of data bits:\t";
    cin >> dataBits;

    vector<int> data(dataBits);
    cout << endl << "NOTE: Make sure the bits are entered in binary format, separated by spaces." << endl;
    cout << "Enter the data bits (rightmost bit first):\t";
    for (int i = 0; i < dataBits; i++) {
        cin >> data[i];
    }

    vector<int> encoded = encodeData(data);
    cout << endl << "--------------------" << endl;
    cout << "Encoded bits are (left to right, position 1 to n):" << endl;
    for (int i = 0; i < encoded.size(); i++) {
        if (i + 1 == pow(2, static_cast<int>(log2(i + 1)))) {
            cout << "p" << (i + 1) << "=" << encoded[i] << " ";
        } else {
            cout << "d" << (i + 1) << "=" << encoded[i] << " ";
        }
    }
    cout << endl << "--------------------" << endl;
    cout << endl << "Enter the encoded bits for error detection (left to right, position 1 to n):\t";
    vector<int> receivedEncoded(encoded.size());
    for (int i = 0; i < encoded.size(); i++) {
        cin >> receivedEncoded[i];
    }

    int errorPosition = checkForErrors(receivedEncoded);
    if (errorPosition == 0) {
        cout << "No errors detected." << endl;
    } else {
        cout << "Error detected at position: " << errorPosition << endl;
        // Correct the error
        if (errorPosition <= receivedEncoded.size()) {
            receivedEncoded[errorPosition - 1] = !receivedEncoded[errorPosition - 1];
            cout << "Corrected encoded bits: ";
            for (int bit : receivedEncoded) {
                cout << bit << " ";
            }
            cout << endl;
        }
    }

    return 0;
}
```

Output:

```
Enter the number of data bits:  4

NOTE: Make sure the bits are entered in binary format, separated by spaces.
Enter the data bits (rightmost bit first):      1 0 1 1

--------------------
Encoded bits are (left to right, position 1 to n):
p1=0 p2=1 d3=1 p4=0 d5=0 d6=1 d7=1
--------------------

Enter the encoded bits for error detection (left to right, position 1 to n):    0 1 1 0 1 1 1
Error detected at position: 5
Corrected encoded bits: 0 1 1 0 0 1 1
```

CRC CODE

```cpp
#include <iostream>
using namespace std;
// Function to perform XOR division for CRC calculation
void performDivision(int temp[], int fs, int gs, int g[]) {
    for (int i = 0; i < fs; i++) {
        if (temp[i] == 1) {  // Only divide if MSB is 1
            for (int j = 0; j < gs; j++) {
                temp[i + j] = temp[i + j] ^ g[j];
            }
        }
    }
}

int main() {
    int fs, gs;

    // Get Frame from user
    cout << "Enter Frame size: ";
    cin >> fs;
    int f[fs + 20];  // Extra space for appended bits

    cout << "Enter Frame bits (space separated): ";
    for (int i = 0; i < fs; i++) {
        cin >> f[i];
    }

    // Get Generator polynomial from user
    cout << "Enter Generator size: ";
    cin >> gs;
    int g[gs];

    cout << "Enter Generator bits (space separated): ";
    for (int i = 0; i < gs; i++) {
        cin >> g[i];
    }

    // Display input frame and generator
    cout << "\nSender Side:";
    cout << "\nOriginal Frame: ";
    for (int i = 0; i < fs; i++) {
        cout << f[i] << " ";
    }

    cout << "\nGenerator Polynomial: ";
    for (int i = 0; i < gs; i++) {
        cout << g[i] << " ";
    }
```

```
// Append zeros to the frame (equal to generator size - 1)
int rs = gs - 1;
for (int i = fs; i < fs + rs; i++) {
    f[i] = 0;
}

cout << "\nFrame after appending " << rs << " zeros: ";
for (int i = 0; i < fs + rs; i++) {
    cout << f[i] << " ";
}

// Create a temporary array for division
int temp[fs + rs];
for (int i = 0; i < fs + rs; i++) {
    temp[i] = f[i];
}

// Perform CRC division
performDivision(temp, fs, gs, g);

// Extract CRC bits (remainder)
int crc[rs];
for (int i = 0, j = fs; i < rs; i++, j++) {
    crc[i] = temp[j];
}

cout << "\nCRC bits: ";
for (int i = 0; i < rs; i++) {
    cout << crc[i] << " ";
}

// Create transmitted frame (original data + CRC)
int tf[fs + rs];
for (int i = 0; i < fs; i++) {
    tf[i] = f[i];
}
for (int i = fs, j = 0; i < fs + rs; i++, j++) {
    tf[i] = crc[j];
}

cout << "\nTransmitted Frame: ";
for (int i = 0; i < fs + rs; i++) {
    cout << tf[i] << " ";
}

// Simulate transmission with possible error
cout << "\n\nReceiver Side:";
cout << "\nEnter received frame (with possible errors): ";
int received[fs + rs];
```

```
    for (int i = 0; i < fs + rs; i++) {
        cin >> received[i];
    }

    // Check for errors
    for (int i = 0; i < fs + rs; i++) {
        temp[i] = received[i];
    }
    // Perform division on received frame
    performDivision(temp, fs, gs, g);

    // Check remainder
    bool errorDetected = false;
    for (int i = fs; i < fs + rs; i++) {
        if (temp[i] != 0) {
            errorDetected = true;
            break;
        }
    }
    if (!errorDetected) {
        cout << "\nNo errors detected. Frame is correct.";
    } else {
        cout << "\nError detected in the received frame!";
        cout << "\nRemainder: ";
        for (int i = fs; i < fs + rs; i++) {
            cout << temp[i] << " ";
        }
    }
    return 0;
}
```

Output:

```
Enter Frame size: 4
Enter Frame bits (space separated): 1 0 1 1
Enter Generator size: 4
Enter Generator bits (space separated): 1 0 0 1

Sender Side:
Original Frame: 1 0 1 1
Generator Polynomial: 1 0 0 1
Frame after appending 3 zeros: 1 0 1 1 0 0 0
CRC bits: 0 1 0
Transmitted Frame: 1 0 1 1 0 1 0

Receiver Side:
Enter received frame (with possible errors): 1 0 1 1 0 0 0

Error detected in the received frame!
Remainder: 0 1 0
```