Experiment No 5

Name: Pranav Santosh Gore

Roll no 22141214

```
#include <bits/stdc++.h>
using namespace std;

//code
// Function to calculate the parity bits and insert them into the correct positions
void calculateParityBits(vector<int>& data, int r) {
    int option;
    cout << "Enter which Parity method do you want to use\n1. Even Parity \n2. Odd Parity\n\nEnter your choice => ";
    cin >> option;

    for (int i = 0; i < r; i++) {
        int parityPos = pow(2, i); // Calculate parity position (1, 2, 4, 8,...)
        int parity = 0;

        for (int j = 1; j <= data.size(); j++) {
            // Check if the parity bit should include the data bit at position j
            if (j & parityPos) {
                parity ^= data[j - 1]; // XOR the bits covered by the parity position
            }
        }

        if (option == 1) {
            data[parityPos - 1] = parity; // Set the parity bit (Even Parity)
        } else if (option == 2) {
            data[parityPos - 1] = !parity; // Set the parity bit (Odd Parity)
        } else {
            cout << "Enter valid choice! Try again :)" << endl;
            calculateParityBits(data, r);
        }
    }
}

// Function to detect and correct errors in the Hamming code
int detectAndCorrectError(const vector<int>& data, int r) {
    int errorPos = 0;

    for (int i = 0; i < r; i++) {
        int parityPos = pow(2, i);
        int parity = 0;

        for (int j = 1; j <= data.size(); j++) {
            if (j & parityPos) {
                parity ^= data[j - 1]; // XOR the bits for this parity position
            }
        }

        if (parity != 0) {
            errorPos += parityPos; // Sum up the positions where parity is incorrect
        }
    }

    return errorPos;
}

// Function to create the Hamming code with parity bits
vector<int> createHammingCode(const vector<int>& inputData) {
    int m = inputData.size();
    int r = 0;
    // Calculate the number of parity bits needed
    while (pow(2, r) < (m + r + 1)) {
        r++;
    }

    int totalBits = m + r;
/8510  vector<int> data(totalBits, 0);

    // Insert the data bits into the positions excluding parity bits
    for (int i = 0, j = 0; i < totalBits; i++) {
        if ((i + 1) && ((i + 1) & i) != 0) {
            data[i] = inputData[j++];
        }
    }

    // Calculate and insert parity bits
    calculateParityBits(data, r);

    return data;
}

int main() {
    cout<<"         **** Implementation Of Hamming Code ****\n"<<endl;
    // Example input data (4 bits of actual data)
    string data;
    cout << "Enter input data: ";
    cin >> data;

    int n = data.length();
    vector<int> inputData(n);
```

```
for (int i = 0; i < n; i++) {
   int num = data[i] - '0';
   inputData[i] = num;
}

reverse(inputData.begin(),inputData.end());

// Create Hamming code
vector<int> hammingCode = createHammingCode(inputData);
vector<int> h=hammingCode;
reverse(hammingCode.begin(),hammingCode.end());
cout << "Hamming code generated: ";
for (int bit : hammingCode) {
   cout << bit;
}
cout << endl;

int pos = 0;
cout << "Enter position of bit you want to change: ";
cin >> pos;

if (pos >= 1 && pos <= hammingCode.size()) {
   // Simulate an error at the chosen position
   hammingCode[pos - 1] = !hammingCode[pos - 1];
   cout << "Hamming code with error: ";

   for (int bit : hammingCode) {
      cout << bit;
   }
   cout << endl;
   reverse(hammingCode.begin(),hammingCode.end());
```

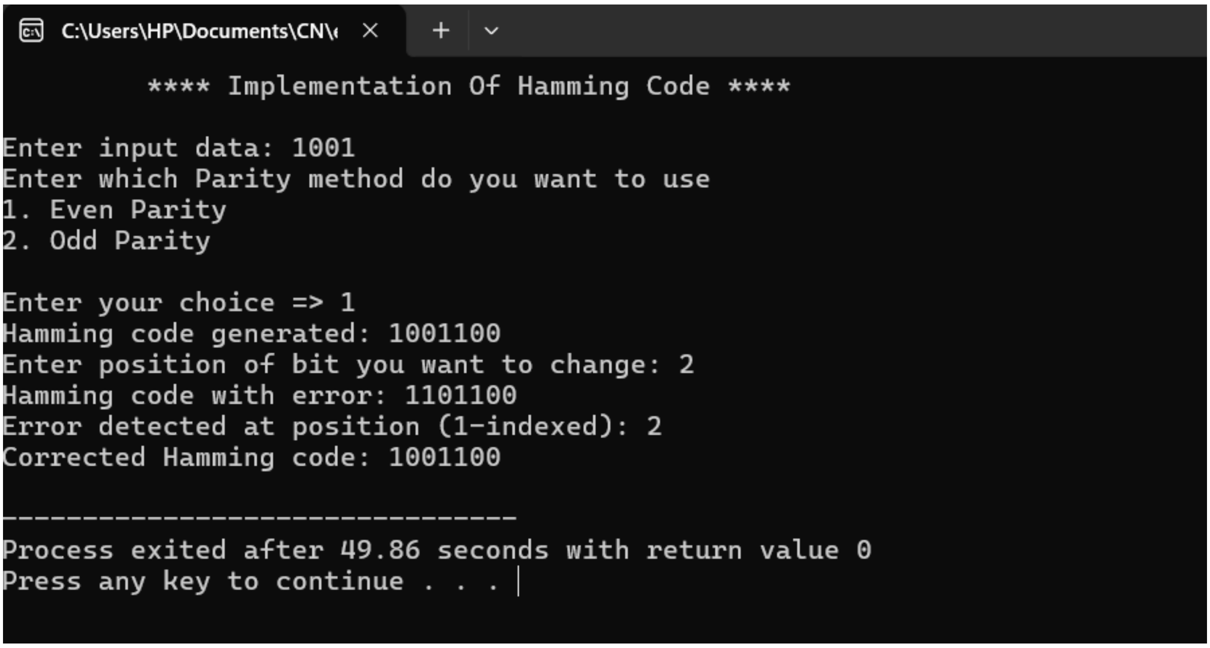
```
        // Detect and correct the error
        int r = log2(hammingCode.size()) + 1;  // Calculate the number of parity bits based on the code size
        int errorPos = detectAndCorrectError(hammingCode, r);

        if (errorPos == 0) {
            cout << "No errors detected." << endl;
        } else {
            cout << "Error detected at position (1-indexed): " << hammingCode.size()-errorPos+1 << endl;
            hammingCode[errorPos - 1] = !hammingCode[errorPos - 1]; // Correct the error

            reverse(hammingCode.begin(),hammingCode.end());
            cout << "Corrected Hamming code: ";
            for (int bit : hammingCode) {
                cout << bit;
            }
            cout << endl;
        }
    } else {
        cout << "Please enter a valid position!" << endl;
    }

    return 0;
}
```

Output:



```
         **** Implementation Of Hamming Code ****

Enter input data: 1001
Enter which Parity method do you want to use
1. Even Parity
2. Odd Parity

Enter your choice => 1
Hamming code generated: 1001100
Enter position of bit you want to change: 2
Hamming code with error: 1101100
Error detected at position (1-indexed): 2
Corrected Hamming code: 1001100

--------------------------------
Process exited after 49.86 seconds with return value 0
Press any key to continue . . .
```