**Lab Experiment – 2**

1. Write a program to implement Error Detection and Correction Technique using Hamming code.

#include <iostream>

#include <cmath>

#include <string>

using namespace std;

bool isInteger(double number)

{

    return floor(number) == number;

}

void display\_hamming\_code(int \*hamming\_code, int total\_count)

{

    for (int i = total\_count - 1; i >= 0; i--)

    {

        cout << hamming\_code[i] << " ";

    }

    cout << endl;

}

int \*generate\_hamming\_code(string data, int &total\_count)

{

    int data\_count = data.length();

    int parity\_count = 1;

    while (pow(2, parity\_count) < data\_count + parity\_count + 1)

    {

        parity\_count += 1;

    }

    total\_count = data\_count + parity\_count;

    int \*hamming\_code = new int[total\_count];

    int data\_ptr = 0;

    for (int i = total\_count; i >= 1; i--)

    {

        if (isInteger(log2(i)))

        {

            hamming\_code[i - 1] = 2;

        }

        else

        {

            hamming\_code[i - 1] = (int)data[data\_ptr++] - 48;

        }

    }

    int count;

    for (int i = 1; i <= total\_count; i = i \* 2)

    {

        count = 0;

        for (int j = i + 1; j <= total\_count; j++)

        {

            if (j & (1 << (int)(log2(i))))

            {

                if (hamming\_code[j - 1])

                {

                    count++;

                }

            }

        }

        if (count % 2 == 0)

        {

            hamming\_code[i - 1] = 0;

        }

        else

        {

            hamming\_code[i - 1] = 1;

        }

    }

    return hamming\_code;

}

int check(int \*hamming\_code, int total\_count)

{

    string bit\_info = "";

    int count;

    for (int i = 1; i <= total\_count; i = i \* 2)

    {

        count = 0;

        for (int j = i + 1; j <= total\_count; j++)

        {

            if (j & 1 << (int)(log2(i)))

            {

                if (hamming\_code[j - 1])

                {

                    count++;

                }

            }

        }

        if (hamming\_code[i - 1])

        {

            count++;

        }

        if (count % 2 == 0)

        {

            bit\_info = "0" + bit\_info;

        }

        else

        {

            bit\_info = "1" + bit\_info;

        }

    }

    return stoi(bit\_info, 0, 2);

}

string retrive\_data\_from\_hammingcode(int \*hamming\_code, int total\_count)

{

    string retrived\_data = "";

    for (int i = total\_count; i >= 1; i--)

    {

        if (isInteger(log2(i)))

        {

            continue;

        }

        else

        {

            retrived\_data += to\_string(hamming\_code[i - 1]);

        }

    }

    return retrived\_data;

}

int main()

{

    string data;

    cout << "Enter the data bits: ";

    cin >> data;

    int total\_count;

    int \*hamming\_code = generate\_hamming\_code(data, total\_count);

    cout << "Generated code is: " << endl;

    display\_hamming\_code(hamming\_code, total\_count);

    cout << "Enter position of a bit to alter, so we can check whether the corruption alogrithm is working at the receivers end or not: " << endl;

    int bit;

    cin >> bit;

    hamming\_code[bit - 1] = hamming\_code[bit - 1] ^ 1;

    cout << "Sent hamming code is: " << endl;

    display\_hamming\_code(hamming\_code, total\_count);

    int corrupted\_bit = check(hamming\_code, total\_count);

    cout << "Error is located at: " << corrupted\_bit << endl;

    hamming\_code[corrupted\_bit - 1] = hamming\_code[corrupted\_bit - 1] ^ 1;

    cout << "Corrected code is : " << endl;

    display\_hamming\_code(hamming\_code, total\_count);

    cout << "Original data sent was: " << endl;

    string retrived\_data = retrive\_data\_from\_hammingcode(hamming\_code, total\_count);

    cout << retrived\_data;

}

Output:

