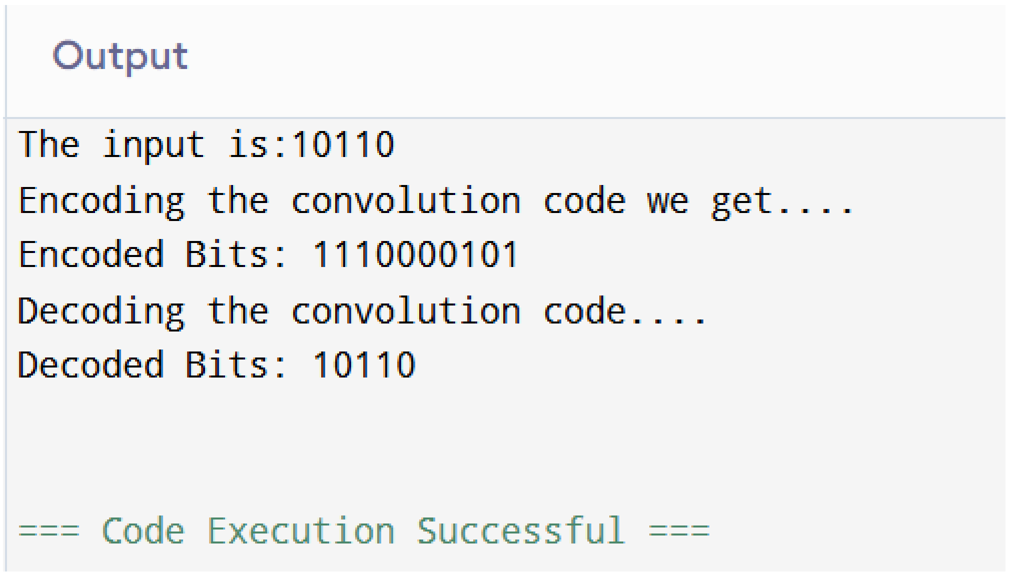
ITC Coding Assignment
Name - Anmol Agrawal
Roll No - 122CS0300

Implementation of Convolution code

```
#include <iostream>
#include <vector>
#include <map>
#include <climits>
using namespace std;

// Encode one step using the generator polynomials
pair<int, int> encodeBits(const vector<int>& shiftRegister) {
    int G1[] = {1, 1, 1}; // 111
    int G2[] = {1, 0, 1}; // 101

    int out1 = 0, out2 = 0;
    for (int i = 0; i < 3; i++) {
        out1 ^= (shiftRegister[i] & G1[i]);
        out2 ^= (shiftRegister[i] & G2[i]);
    }

    return {out1, out2};
}

// Encoding the full input bitstream
vector<int> convolutionalEncode(const vector<int>& inputBits) {
    vector<int> outputBits;
    vector<int> shiftRegister(3, 0);

    for (int bit : inputBits) {
        shiftRegister[2] = shiftRegister[1];
        shiftRegister[1] = shiftRegister[0];
        shiftRegister[0] = bit;
        auto [out1, out2] = encodeBits(shiftRegister);
        outputBits.push_back(out1);
        outputBits.push_back(out2);
    }

    return outputBits;
}

// Viterbi Decoder for the given encoded sequence
vector<int> viterbiDecode(const vector<int>& received) {
    const int numStates = 4;
    const int numSteps = received.size() / 2;
    vector<vector<int>> paths(numStates);
    vector<int> pathMetric(numStates, INT_MAX);
    pathMetric[0] = 0;

    for (int t = 0; t < numSteps; ++t) {
        vector<int> newMetric(numStates, INT_MAX);
        vector<vector<int>> newPaths(numStates);

        int r1 = received[2 * t];
        int r2 = received[2 * t + 1];

        for (int state = 0; state < numStates; ++state) {
            if (pathMetric[state] == INT_MAX) continue;

            for (int inputBit = 0; inputBit <= 1; ++inputBit) {
                int prev0 = (state >> 1) & 1;
                int prev1 = state & 1;

                vector<int> shiftRegister = {inputBit, prev0, prev1};
                auto [e1, e2] = encodeBits(shiftRegister);

                int nextState = ((inputBit << 1) | prev0);
                int dist = (e1 != r1) + (e2 != r2); // Hamming distance

                if (pathMetric[state] + dist < newMetric[nextState]) {
                    newMetric[nextState] = pathMetric[state] + dist;
                    newPaths[nextState] = paths[state];
                    newPaths[nextState].push_back(inputBit);
                }
            }
        }

        pathMetric = newMetric;
        paths = newPaths;
    }
    int minMetric = INT_MAX, bestState = 0;
    for (int state = 0; state < numStates; ++state) {
        if (pathMetric[state] < minMetric) {
            minMetric = pathMetric[state];
            bestState = state;
        }
    }
    return paths[bestState];
}

int main() {
    vector<int> inputBits = {1, 0, 1, 1, 0};
    cout<<"The input is:"<<10110<<endl;
    cout<<"Encoding the convolution code we get...."<<endl;
    vector<int> encoded = convolutionalEncode(inputBits);

    cout << "Encoded Bits: ";
    for (int bit : encoded) cout << bit;
    cout << "\n";
    cout<<"Decoding the convolution code...."<<endl;
    vector<int> decoded = viterbiDecode(encoded);
```

```
    cout << "Decoded Bits: ";
    for (int bit : decoded) cout << bit;
    cout << "\n";

    return 0;
}
```

<u>OUTPUT:</u>



Output

```
The input is:10110
Encoding the convolution code we get....
Encoded Bits: 1110000101
Decoding the convolution code....
Decoded Bits: 10110


=== Code Execution Successful ===
```