422116—WEEK1

MY Details

NAME : **B. THARUN**

ROLL NO : **422116**

SECTION : **A**

LAB-1 (02-01-2025)

1. Construct a Program for Bits Stuffing(A Bits stuffing is a program where if we get five consecutive one's then our output should be five consecutive one's we have to insert 0)

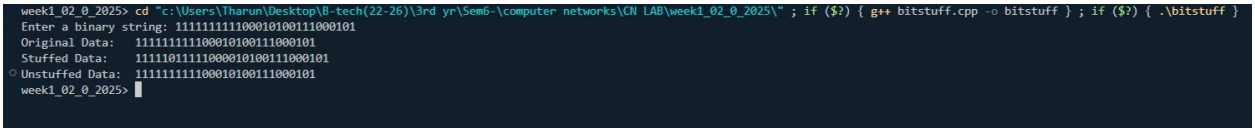
Input: 111011111111101111110

Output: 11101111101111011111010

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


vector<int> to_int(string s) {
    vector<int> v;
    for (char c : s) {
        if (c == '0' || c == '1') {
            v.push_back(c - '0');
        } else {
            cerr << "Invalid input: Only binary digits (0 or
1) are allowed." << endl;
            exit(1); // Exit the program
        }
    }
    return v;
```

```
}

vector<int> bitstuff(vector<int> data) {
    int b1 = 0;
    vector<int> v;
    for (auto i : data) {
        if (i == 1) {
            b1++;
            v.push_back(i);
            if (b1 == 5) { // x11111x case
                v.push_back(0);
                b1 = 0;
            }
        } else {
            v.push_back(i);
            b1 = 0; // Reset consecutive 1s count
        }
    }
    return v;
}

vector<int> unbitstuff(vector<int> data) {
    int b1 = 0;
    vector<int> v;
    for (auto i : data) {
        if (i == 1) {
            b1++;
            v.push_back(i);
        } else {
            if (b1 == 5) {
                // Skip the stuffed 0
                b1 = 0;
                continue;
            }
            v.push_back(i);
            b1 = 0;
```

```
        }
    }
    return v;
}

int main() {
    string s;
    cout << "Enter a binary string: ";
    getline(cin, s);
    vector<int> v = to_int(s);

    vector<int> stuffed = bitstuff(v);
    vector<int> unstuffed = unbitstuff(stuffed);


    cout << "Original Data:   ";
    for (int i : v) cout << i;
    cout << endl;

    cout << "Stuffed Data:    ";
    for (int i : stuffed) cout << i;
    cout << endl;

    cout << "Unstuffed Data:  ";
    for (int i : unstuffed) cout << i;
    cout << endl;

    return 0;
}
```



2. Construct a Program for Cyclic Redundancy Check(Error detection techniques)

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

void XOR(vector<int> &dividend, vector<int> &divisor, int sta
rt)
{
    for (int i = 0; i < divisor.size(); i++)
    {
        dividend[start + i] ^= divisor[i];
    }
}

vector<int> computeCRC(vector<int> data, vector<int> generato
r)
{
    int n = generator.size();
    vector<int> dividend = data;

    for (int i = 0; i < n - 1; i++)
    {
        dividend.push_back(0);
    }

    for (int i = 0; i <= dividend.size() - n; i++)
    {
        if (dividend[i] == 1)
        {
            XOR(dividend, generator, i);
        }
    }

    vector<int> remd(dividend.end() - (n - 1), dividend.end
());
```

```
    return remd;
}

bool verifyCRC(vector<int> rData, vector<int> gen)
{
    int n = gen.size();

    for (int i = 0; i <= rData.size() - n; i++)
    {
        if (rData[i] == 1)
        {
            XOR(rData, gen, i);
        }
    }

    for (int i = rData.size() - (n - 1); i < rData.size(); i+
+)
    {
        if (rData[i] != 0)
        {
            return false;
        }
    }
    return true;
}

int main()
{

    vector<int> data = {1, 0, 1, 1, 1};
    vector<int> generator = {1, 0, 1};

    vector<int> crc = computeCRC(data, generator);

    cout << "Original Data: ";
    for (int bit : data)
```

```
        cout << bit;
    cout << "\nCRC: ";
    for (int bit : crc)
        cout << bit;

    vector<int> transmittedData = data;
    transmittedData.insert(transmittedData.end(), crc.begin
(), crc.end()); // to have error case we can modify that data
which is transmitting to receiver

    cout << "\nTransmitted Data: ";
    for (int bit : transmittedData)
        cout << bit;

    bool isValid = verifyCRC(transmittedData, generator);

    cout << "\nData Verification Result: " << (isValid ? "No
Error" : "Error Detected") << endl;

    return 0;
}
```