IT 205

COMPUTER COMMUNICATION AND NETWORKING – LAB ASSIGNMENT 4

By-

Sachin Prasanna

211IT058

Source Code Screenshots:

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

//BYTE STUFFING CODE

string byteStuffingEncoding(string code){

    vector <char> encode;
    encode.push_back('0');
    encode.push_back('2');
    encode.push_back(' ');
    encode.push_back('1');
    encode.push_back('6');
    encode.push_back(' ');
    for(int i = 0 ; i<code.size() ; i++){
        encode.push_back(code[i]);
        if(encode[encode.size() - 2] == '6' && encode[encode.size() - 3] == '1'){
            encode.push_back('1');
            encode.push_back('6');
            encode.push_back(' ');
        }
        if((i == code.size() - 1) && encode[encode.size() - 1] == '6' && encode[encode.size() - 2] == '1'){
            encode.push_back(' ');
            encode.push_back('1');
            encode.push_back('6');
        }
    }
    encode.push_back(' ');
    encode.push_back('1');
    encode.push_back('6');
    encode.push_back(' ');
    encode.push_back('0');
    encode.push_back('1');
    string final(encode.begin(), encode.end());
    return final;
}

string byteStuffingDecoding(string code){

    vector <char> decode;
    int i = 6;
    while (i < code.size() - 6){
        decode.push_back(code[i]);
        if(decode[decode.size() - 1] == '6' && decode[decode.size() - 2] == '1'){
            i = i + 4;
        }
        else
            i++;
    }
    string final(decode.begin(), decode.end());
    return final;
}

int overheadBytes(string code, string ans){

    int ct1= 0;
    int ct2 = 0;
    for(int i = 0; i < code.size(); i++){
        if(code[i] == ' ')
            ct1++;
    }
```

```
65
66      for(int i = 0; i < ans.size(); i++){
67          if(ans[i] == ' ')
68              ct2++;
69      }
70
71      return (ct2 - ct1);
72  }
73
74  //BIT STUFFING CODE
75
76  string hexadecimalToBinary(string hexadecimal){
77      string answer = "";
78      for(int i=0;i<2;i++){
79      if(hexadecimal[i] == '\0' || hexadecimal[i] == ' '){
80          cout<<"Numbers inputted should be 1 byte"<<endl;
81          return "";
82      }
83
84      switch(hexadecimal[i])
85          {
86              case '0':
87                  answer.append("0000");
88                  break;
89              case '1':
90                  answer.append("0001");
91                  break;
92              case '2':
93                  answer.append("0010");
94                  break;
95              case '3':
96                  answer.append("0011");
97                  break;
98              case '4':
99                  answer.append("0100");
100                 break;
101             case '5':
102                 answer.append("0101");
103                 break;
104             case '6':
105                 answer.append("0110");
106                 break;
107             case '7':
108                 answer.append("0111");
109                 break;
110             case '8':
111                 answer.append("1000");
112                 break;
113             case '9':
114                 answer.append("1001");
115                 break;
116             case 'A':
117                 answer.append("1010");
118                 break;
119             case 'B':
120                 answer.append("1011");
121                 break;
122             case 'C':
123                 answer.append("1100");
124                 break;
125             case 'D':
126                 answer.append("1101");
127                 break;
128             case 'E':
```

```
                answer.append("1110");
                break;
            case 'F':
                answer.append("1111");
                break;
            default:
                cout<<"Enter Hexadecimal characters!";
        }
    }
    return answer;
}

string BitStuffingEncoding(string code){
    int ct=0;
    for(int i=0;i<code.length();i++){
        if(code[i] == '1'){
            ct++;
            if(ct == 5){
                code.insert(i+1,"0");
            }
        }
        else
            ct=0;
    }
    return code;
}

vector<string> encodePackets(vector<string> binaryData){
    for(int i=0;i<binaryData.size();i++){
        binaryData[i] = BitStuffingEncoding(binaryData[i]);
    }
    return binaryData;
}
```

```
string header_trailer = "01111110";

string BitStuffingEncode(vector<string> Data_bin){
    string Data_binStr = "";
    for(auto i: Data_bin){
        Data_binStr.append(i);
        Data_binStr.append(" ");

    }
    return header_trailer+ "  " +Data_binStr+ " " +header_trailer;
}


int overheadBitsStuffing(vector<string> data){

    int count = 16;
    for(int i = 0; i < data.size() ; i++){
        if(data[i].size() == 9)
            count++;
    }

    return count ;
}


string BitStuffingDecoding(string code){
    int count=0;
    string ans = "";
    for(int i=0;i<code.length();i++){
        if(code[i] == '1'){
```

```
            count++;
            ans = ans+ code[i];
            if(count == 5){
                i++;
            }
        }else if(code[i] == '0'){
            count = 0;
            ans = ans+ code[i];
        }
    }
    return ans;
}

string decodePackets(string code){
    code = code + " ";
    string input;
    string output;
    for(int i=0;i<code.length();i++){
        if(code[i] == ' '){
            output.append(BitStuffingDecoding(input));
            output.append(" ");
            input.clear();
        }else{
            input = input + code[i];
        }
    }
    return output;
}

string frameSlice(string frame){
    return frame.substr(10,frame.length()-8-2-8-2);
}
```

```

void display(vector<string> inp){
    for(int i=0;i<inp.size();i++){
        cout<<inp[i]<<" ";
    }
    cout<<endl;
}

int main(){

    //Byte Stuffing

    string code;
    cout<<"*****BYTE STUFFING AND BIT STUFFING*****\n\n";
    cout<<"Enter your CODE in hexadecimal, one by one which will be inputted into the frame: ";
    getline(cin, code);

    cout<<"\n*****PERFORMING BYTE STUFFING*****\n\n";

    cout<<"INPUTTED CODE (Data to be sent): ";
    cout<<code;

    string ans = byteStuffingEncoding(code);

    cout<<"\n\nENCODED CODE (Frames which are sent by the sender to the receiver): "<<ans;

    cout<<"\n\nNumber of overhead Bytes according to Byte Stuffing is : "<<overheadBytes(code,ans);
    cout<<"\nNumber of overhead Bits according to Byte Stuffing is: "<<overheadBytes(code,ans)*8;

    string decode = byteStuffingDecoding(ans);

    cout<<"\n\nDECODED CODE (Data recieved by the reciever): "<<decode;
```

```
257     cout<<"\n\nClearly Inputted Code and Decoded Code are the SAME according to Byte Stuffing!\n\n\n";
258
259     //Bit Stuffing
260
261     cout<<"*****PERFORMING BIT STUFFING*****"<<endl;
262     vector<string> binaryData;
263     int i = 0;
264     while(i<code.length()){
265         string binary = hexadecimalToBinary(code.substr(i,2));
266         if(binary == ""){
267             return -1;
268         }
269         binaryData.push_back(binary);
270         i+=3;
271     }
272     cout<<"INPUTTED CODE: ";
273     display(binaryData);
274     vector<string> Data_bin = encodePackets(binaryData);
275
276     string frame = BitStuffingEncode(Data_bin);
277     cout<<"\n\nENCODED CODE: "<<frame;
278
279     cout<<"\n\nNumber of overhead Bits according to Bit Stuffing is: "<<overheadBitsStuffing(Data_bin);
280     string data_final = frameSlice(frame);
281     string output = decodePackets(data_final);
282     cout<<"\n\nDECODED CODE : "<<output<<endl;
283
284     cout<<endl<<endl;
285
286     cout<<"Clearly Inputted and Decoded Code are the SAME according to Bit Stuffing!";
287     return 0;
288 }
```

Output Screenshot:

```
input
*****BYTE STUFFING AND BIT STUFFING*****

Enter your CODE in hexadecimal, one by one which will be inputted into the frame: 01 04 FF 16 01 16 0F F4 15 16 08

*****PERFORMING BYTE STUFFING*****

INPUTTED CODE (Data to be sent): 01 04 FF 16 01 16 0F F4 15 16 08

ENCODED CODE (Frames which are sent by the sender to the receiver): 02 16 01 04 FF 16 16 01 16 16 0F F4 15 16 16 08 16 01

Number of overhead Bytes according to Byte Stuffing is : 7
Number of overhead Bits according to Byte Stuffing is: 56

DECODED CODE (Data recieved by the reciever): 01 04 FF 16 01 16 0F F4 15 16 08

Clearly Inputted Code and Decoded Code are the SAME according to Byte Stuffing!


*****PERFORMING BIT STUFFING*****
INPUTTED CODE: 00000001 00000100 11111111 00010110 00000001 00010110 00001111 11110100 00010101 00010110 00001000


ENCODED CODE: 01111110  00000001 00000100 111110111 00010110 00000001 00010110 00001111 11110100 00010101 00010110 00001000
01111110

Number of overhead Bits according to Bit Stuffing is: 17

DECODED CODE : 00000001 00000100 11111111 00010110 00000001 00010110 00001111 11110100 00010101 00010110 00001000


Clearly Inputted and Decoded Code are the SAME according to Bit Stuffing!

...Program finished with exit code 0
Press ENTER to exit console.
```

Observations:

Implemented Byte Stuffing and Bit Stuffing by writing a program in C++.

Bit Stuffing and Byte Stuffing let the receiver know about the size of the frame by using different techniques. (i.e. Adding extra Bit for Bit Stuffing and extra Byte for Byte Stuffing)

Given Input in hexadecimal: 01 04 FF 16 01 16 0F F4 15 16 08

Byte Stuffing

Since the given DLE is 16, whenever 16 is encountered in the input, an extra 16 is stuffed into the input. This helps when the receiver decodes the code and knows to remove one 16 when two simultaneous 16s are encountered.

In the given Input, there are 3 16s, so another 3 16s will be stuffed into the input. The STX, DLE, DLE, ETX are 1 byte each and 4 bytes in total and they are also stuffed into the input. Since each 16 is a byte and 8 bits, the total number of stuffed bytes or Overhead bytes = 4+3 = 7. Correspondingly, the number of stuffed bits will be 7*8 = 56, which is shown in the output. (1 byte = 8 bits)

Also, STX and DLE are stuffed into the starting of the input and DLE and ETX are stuffed into the ending of the input. Finally, this frame is sent to the receiver.

Upon receiving, the receiver removes 16 if there are 2 16s together. This gets back the original sent data.

This is the process of Byte Stuffing.

Bit Stuffing

Since Bit Stuffing works on Bits, the given hexadecimal input is converted into Bits.

The header and trailer will be same in Bit Stuffing unlike Byte Stuffing, and the given sequence for the same is 01111110.

As we traverse through each byte, if any byte has 5 1s in a row, then we add an extra 0 to it to differentiate from the starting and ending sequence.

The header and trailer are 16 bits each, so by default the overhead Bits will be 16 or more. Then, since only one byte has 5 1s in a row, the number of Overhead Bits = 16+1 = 17.

Also, the sequence is stuffed into the starting and ending of the input and sent to the receiver for decoding.

Upon receiving, if the receiver notices 5 1s in a row, apart from starting and ending, it removes the extra 0 after it, as it knows it has been added extra by the sender. The original data is thus retrieved.

This is the process of Bit Stuffing.

THANK YOU