Green University of Bangladesh
Department of Computer Science and Engineering(CSE)

Faculty of Sciences and Engineering
Semester: (Spring, Year:2024), B.Sc. in CSE (Day)

LAB REPORT NO #01
Course Title: Data Communication Lab
Course Code: CSE 308 Section: 221_D3

Lab Experiment Name:

1. **Implementing Byte (Character) Stuffing and De-stuffing,**
2. **Implementing Bit Stuffing and De-stuffing and**
3. **IPv4 implementation of Decimal to Binary and vice versa**

Student Details

	Name	ID
1.	Jahidul Islam	221002504

Lab Date : 02 – 03 – 2024
Submission Date : 16 – 03 – 2024
Course Teacher's Name : Sakhaouth Hossan

[For Teachers use only: Don't Write Anything inside this box]

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

1. TITLE OF THE LAB EXPERIMENT:

a) **Implementing Byte (Character) Stuffing and De-stuffing**
b) **Implementing Bit Stuffing and De-stuffing.**
c) **IPv4 implementation of Decimal to Binary and vice versa.**

2. OBJECTIVES

After complementing this lab experiment, we will gain practical knowledge and tthe outcomes of this experiment are

1. Bit stuffing
2. Bit destuffing
3. Byte stuffing and de-stuffing
4. Decimal to Binary conversion of Ipv4 number.

3. PROCEDURE

a) Byte (Character) Stuffing and De-stuffing:

- Problem Understanding:
  - The problem is about implementing a byte stuffing algorithm in C++.
  - the FLAG sequence ("GALF") and the ESCAPE sequence ("EPACSE").

- Algorithm:
  - Implementation of the algorithm as a function that takes the input data, FLAG sequence, and ESCAPE sequence as parameters and returns the stuffed data.
- Coding:
  - Write the C++ code implementing the byte stuffing algorithm.
  - Use standard input/output for user interaction.

b) Implementing Bit Stuffing and De-stuffing:

1. **Bit Stuffing Procedure:**
   For each bit in the input stream:
   a. Append the bit to the stuffed stream.
   b. If five consecutive '1's are encountered, add an extra '0' to the stuffed stream.

2. **Bit De-stuffing Procedure:**
   For each bit in the stuffed stream:
   a. Append the bit to the de-stuffed stream.
   b. Skip an extra '0' following five consecutive '1's.

c) IPv4 implementation of Decimal to Binary and vice versa

- **IPv4 Decimal to Binary Conversion:**
  1. **Input**: Obtain the decimal IPv4 address to be converted.
  2. **Split**: Split the decimal IPv4 address into four octets separated by periods.
  3. **Conversion (for each octet):**
     - Convert each decimal octet to its 8-bit binary representation.
     - Use bitwise operations or library functions like std::bitset to perform the conversion.
     - Ensure that each binary octet is represented as an 8-bit binary number.
  4. **Concatenation:** Concatenate the binary representations of all octets to form the complete 32-bit binary IPv4 address.

- **IPv4 Binary to Decimal Conversion:**
  1. **Input:** Obtain the binary IPv4 address to be converted.
  2. **Split:** Split the binary IPv4 address into four octets separated by periods.
  3. **Conversion (for each octet):**
     - Convert each 8-bit binary octet to its decimal representation.
     - Use bitwise operations or library functions like std::bitset to perform the conversion.
     - Ensure that each decimal octet is represented as a decimal number between 0 and 255.
  4. **Concatenation:** Concatenate the decimal representations of all octets to form the complete IPv4 address in dotted decimal notation.

4. IMPLEMENTATION

a) Implementation of Byte (Character) Stuffing and De-stuffing:

```
// Bismillahir Rahmanir Rahim
// jahidulZaid
// byteStuffingAlgorithm.
#include <bits/stdc++.h>
using namespace std;
```

```cpp
#define optimize()
ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'
#define tt long long t; cin >> t;
#define ll long long
#define pb push_back

string byteStuffing(const string& data, const string& FLAG, const
string& ESCAPE) {
    string stuffedData;
    stuffedData += FLAG;

    for (size_t i = 0; i < data.length(); ++i) {
        if (data.substr(i, FLAG.length()) == FLAG || data.substr(i,
ESCAPE.length()) == ESCAPE) {
            stuffedData += ESCAPE;
        }
        stuffedData += data[i];
    }
    stuffedData += FLAG;
    return stuffedData;
}
int main() {
    string FLAG = "GALF";
    string ESCAPE = "EPACSE";

    string data;
    getline(cin, data);

    string stuffedData = byteStuffing(data, FLAG, ESCAPE);
    cout << stuffedData << endl;

    return 0;
}
```

b) Implementing Bit Stuffing and De-stuffing:

```cpp
// Bismillahir Rahmanir Rahim
// jahidul Islam
// 221002504

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

```
#define optimize() ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'
#define tt long long t; cin >> t;
#define ll long long
#define pb push_back
#define ppb pop_back

void destuf(string input){
    int startFlag = input.find("01111110");
    int endFlag = input.rfind("01111110");
    string stuffedData = input.substr(startFlag + 8, endFlag - startFlag - 8);
    int size = stuffedData.size();
    vector<char> v;
    int cnt = 0;
    for (int i = 0; i < size; i++) {
        v.push_back(stuffedData[i]);
        if (stuffedData[i] == '1') {
            cnt++;
            if (cnt == 5) {
                i++;
                cnt = 0;
            }
        } else {
            cnt = 0;
        }
    }
    // Print the de-stuffed data with the flags
    cout << "01111110";
    for (auto x : v) {
        cout << x;
    }
    cout << "01111110" << endl;
}
void stuffing(string input) {
    int startFlag = input.find("01111110");
    int endFlag = input.rfind("01111110");
    string destuffedData = input.substr(startFlag + 8, endFlag - startFlag -
8);    vector<char> stuffedData;
    int count = 0;
    for (char c : destuffedData) {
        stuffedData.push_back(c);
        if (c == '1') {
            count++;
            if (count == 5) {
                stuffedData.push_back('0');
```

```
                count = 0;
            }
        } else {
            count = 0;
        }
    }
    // Print the stuffed data with the flags
    cout << "01111110";
    for (char c : stuffedData) {
        cout << c;
    }
    cout << "01111110" << endl;
}
int main() {
    optimize();
    string stuff;
    getline(cin, stuff);
    string input;
    getline(cin, input);
    stuffing(stuff);
    destuf(input);
    return 0;
}
```

c) IPv4 implementation of Decimal to Binary and vice versa

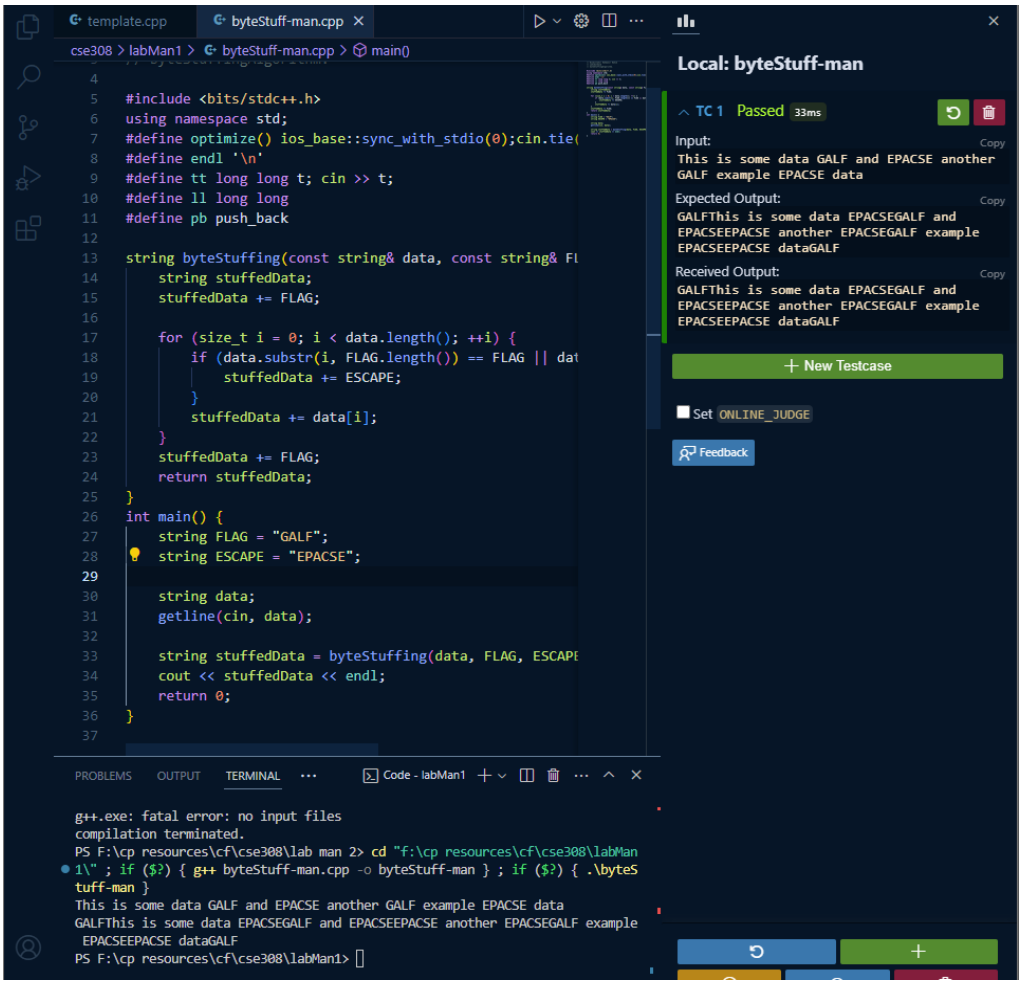
```
// Bismillahir Rahmanir Rahim
// jahidulZaid
#include <bits/stdc++.h>
using namespace std;
#define optimize() ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'
#define tt long long t; cin >> t;
#define ll long long
#define pb push_back

// #ifdef LOCAL
// #include "debug.h"
// #endif

// #ifdef ONLINE_JUDGE
// #include "debug.h"
// #endif
```

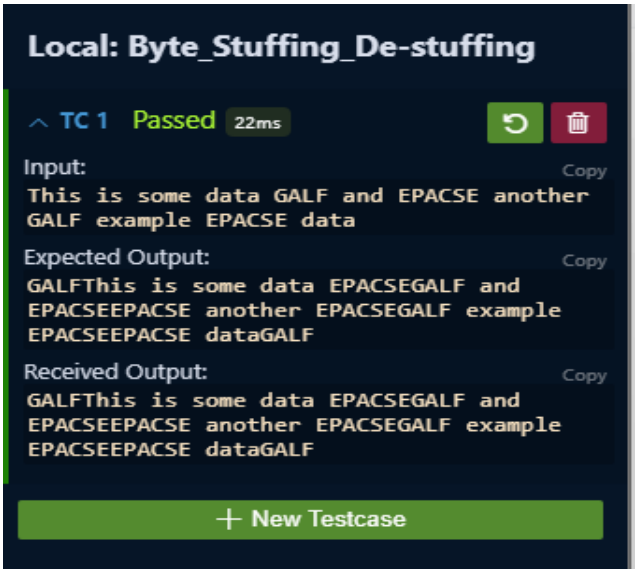
```
char checkIPAddressClass(const string& ipAddress) {
    // Split by '.'
    stringstream ss(ipAddress);
    string octet;
    getline(ss, octet, '.');
    int firstOctet = stoi(octet);

    if (firstOctet >= 1 && firstOctet <= 126)
        return 'A';
    else if (firstOctet >= 128 && firstOctet <= 191)
        return 'B';
    else if (firstOctet >= 192 && firstOctet <= 223)
        return 'C';
    else
        return '0';
}


string binaryToDecimal(const string& binaryIP) {
    string decimalIP = "";
    for (int i = 0; i < 32; i += 8) {
        string octet = binaryIP.substr(i, 8);
        bitset<8> bits(octet);
        decimalIP += to_string(bits.to_ulong());
        decimalIP += ".";
    }
    // Remove the last '.'
    decimalIP.pop_back();
    return decimalIP;
}
string decimalToBinary(const string& decimalIP) {
    stringstream ss(decimalIP);
    string octet;
    string binaryIP = "";
    while (getline(ss, octet, '.')) {
        int value = stoi(octet);
        bitset<8> bits(value);
        binaryIP += bits.to_string();
        binaryIP += ".";
    }
    // Remove the last '.'
    binaryIP.pop_back();
    return binaryIP;
}
char inputIPAddressClass() {
    cout << "Enter the IP address class (A, B, or C): ";
    char ipClass;
    cin >> ipClass;
    ipClass = toupper(ipClass);
    if (ipClass != 'A' && ipClass != 'B' && ipClass != 'C') {
```

```
        cout << "Invalid IP address class." << endl;
        return '0';
    }
    return ipClass;
}
int main() {
    char ipClass = inputIPAddressClass();

    if (ipClass == '0') {
        return 1;
    }
    string ipAddress;
    cout << "Enter an IP address: ";
    cin >> ipAddress;

    cout << "Options:" << endl;
    cout << "1. Convert to binary" << endl;
    cout << "2. Convert from binary" << endl;
    cout << "Enter your choice (1 or 2): ";

    int choice;
    cin >> choice;

    if (choice == 1) {
        string binaryIP = decimalToBinary(ipAddress);
        cout << "Binary IP address: " << binaryIP << endl;
    } else if (choice == 2) {
        string decimalIP = binaryToDecimal(ipAddress);
        cout << "Decimal IP address: " << decimalIP << endl;
    } else {
        cout << "Invalid choice." << endl;
    }

    return 0;
}
```

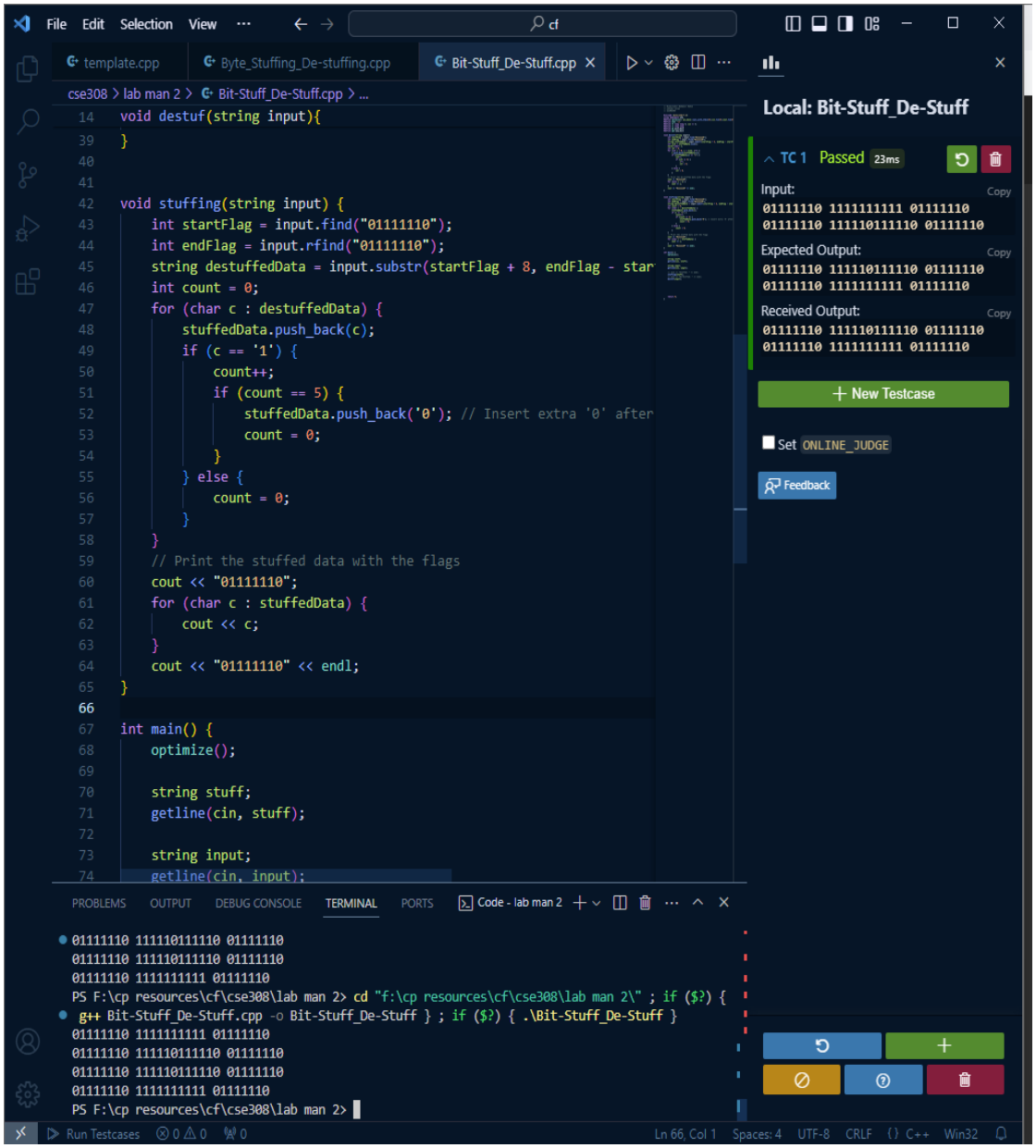
5. OUTPUT

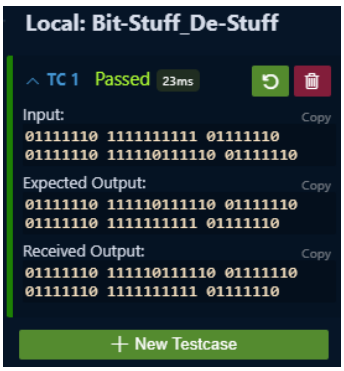
a) Byte (Character) Stuffing and De-stuffing:



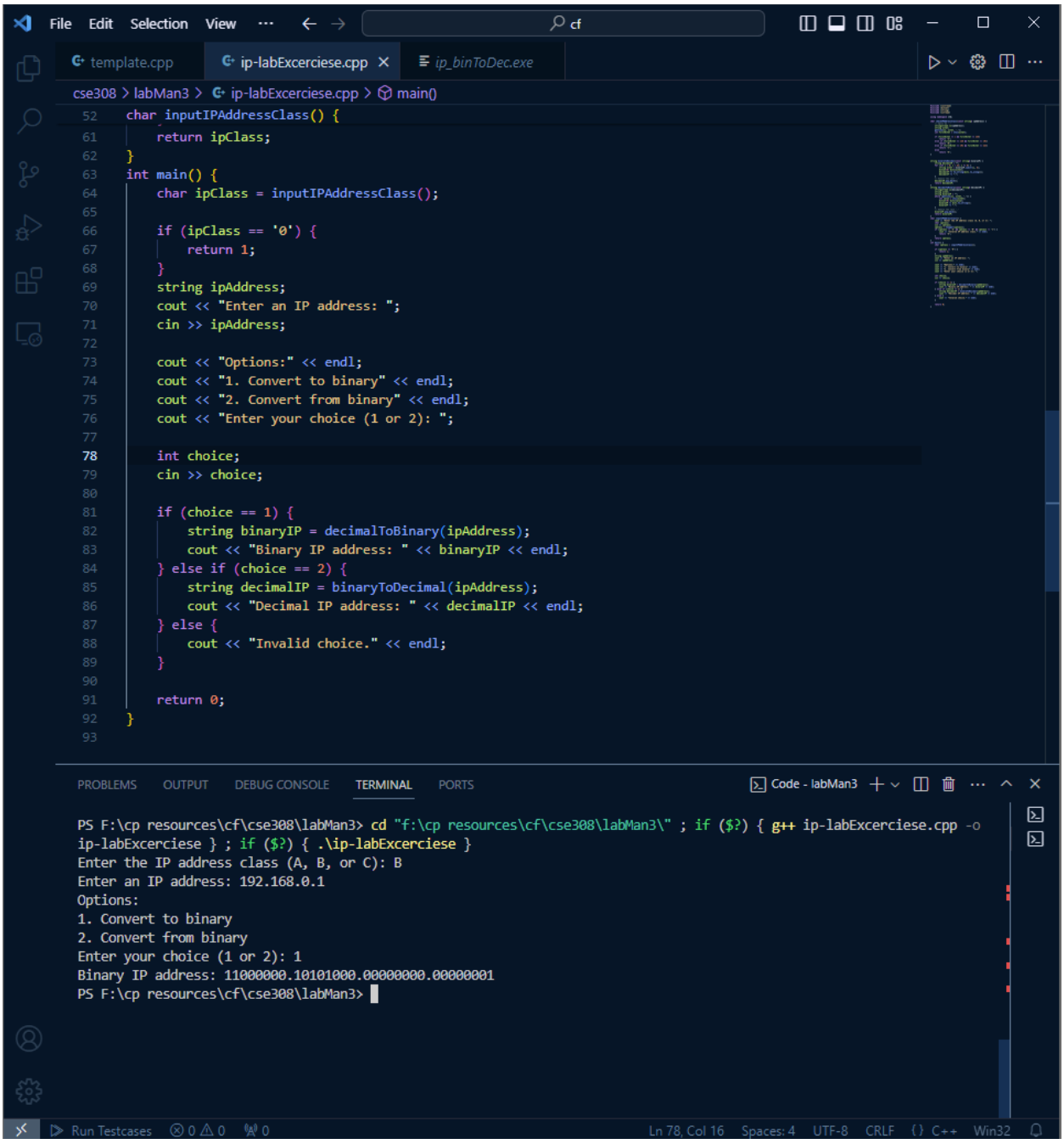


b) Implementing Bit Stuffing and De-stuffing:





c) IPv4 implementation of Decimal to Binary and vice versa



6. ANALYSIS AND DISCUSSION:

a) Byte (Character) Stuffing and De-stuffing:

The byte stuffing algorithm is a fundamental technique used in data communication to ensure reliable transmission of data where certain byte sequences have special meanings and need to be escaped.
In this problem, we implemented a byte stuffing algorithm in C++ according to a specified protocol, which involves inserting an ESCAPE sequence before any occurrence of the FLAG or ESCAPE sequence itself.

The algorithm works by iterating through the input data and inserting the ESCAPE sequence before each occurrence of the FLAG or ESCAPE sequence. This ensures that these special sequences are correctly interpreted during transmission and do not interfere with the data stream's integrity.

b) Byte (Character) Stuffing and De-stuffing:

Byte stuffing and destuffing are techniques used in data communication to ensure the integrity and synchronization of transmitted data, especially in the presence of reserved or control characters that may be misinterpreted as control signals by the receiving system. These techniques are commonly employed in protocols like HDLC (High-Level Data Link Control) and PPP (Point-to-Point Protocol).

- Efficiency: Byte stuffing and destuffing introduce overhead in terms of extra bits or bytes added to the transmitted data. This overhead reduces the efficiency of data transmission.
- Synchronization: These techniques help maintain synchronization between the sender and receiver by ensuring that the receiver can correctly identify the start and end of frames.

- Error Detection: Byte stuffing and destuffing help in error detection by making it easier to distinguish between data and control characters.
- Implementation Complexity: Implementing byte stuffing and destuffing algorithms requires careful handling of special characters, which can increase the complexity of the communication protocol implementation.

c) IPv4 implementation of Decimal to Binary and vice versa

converting between decimal and binary formats is crucial for various network operations, such as routing, subnetting, and configuring network devices.

1. **IPv4 Address Representation:**
    - IPv4 addresses are represented in dotted decimal notation, where each decimal number represents an octet (8 bits) of the IPv4 address, separated by periods.
2. **Conversion Algorithm:**
    - To convert a decimal IPv4 address to binary, each decimal octet needs to be converted to its 8-bit binary representation.
    - This can be achieved by converting each decimal octet to binary using bitwise operations or by using library functions like std::bitset.
3. **Implementation Steps:**
    - Iterate through each decimal octet of the IPv4 address.
    - Convert each decimal octet to its 8-bit binary representation.
    - Concatenate the binary representations of all octets to form the complete 32-bit binary IPv4 address.
4. **Example:**
    - Decimal IPv4 Address: 192.168.1.1
    - Binary Representation: 11000000.10101000.00000001.00000001

Implementing decimal to binary and binary to decimal conversions for ipv4 addresses is essential for network operations and requires accurate, efficient, and robust algorithms to handle various scenarios effectively.

7. SUMMARY:

a. **Implementing Byte (Character) Stuffing and De-stuffing:**
   - Byte stuffing adds extra bits or bytes to data to distinguish them from control characters, ensuring data integrity during transmission.
   - Special characters like the "flag" and "escape" are defined to mark the beginning and end of frames and escape control characters, respectively.
   - Byte destuffing involves detecting and removing escape characters at the receiver's end to recover the original data.

b. **Implementing Bit Stuffing and De-stuffing:**
   - Bit stuffing adds extra bits to data to prevent unintended interpretations of control characters, using a predefined pattern (e.g., five consecutive 1s) as the flag.
   - The sender inserts a bit of the opposite polarity whenever it encounters the flag pattern in the data, and the receiver removes the stuffed bits to restore the original data.
   - Bit stuffing and destuffing ensure synchronization and data integrity in synchronous communication protocols.

c. **IPv4 Implementation of Decimal to Binary and vice versa:**
   - Decimal to binary conversion involves splitting the decimal IPv4 address into octets and converting each octet to its 8-bit binary representation.
   - Binary to decimal conversion requires splitting the binary IPv4 address into octets and converting each octet to its decimal representation.
   - These procedures are crucial for various network operations, such as routing, subnetting, and configuring network devices, ensuring accurate representation and manipulation of IPv4 addresses.