EXP-5: Implement Framing Mechanism using any Programming Language

I. Aim:

Implement Framing Mechanism using Python Programming Language

II. Components and Tools:

System: Desktop Computer/Laptop

Operating system: Windows/Linux

Language: Python

III. Description (Problem Statement):

Task 5.1: Character Stuffing and Destuffing

Task-5.2: Bit Stuffing and Destuffing

IV. Algorithm:

V. Program:

VI. Output Description/Analysis:

VII. Viva-Voce Questions:

Description (Problem Statement)

Task 5.1: Character Stuffing and Destuffing

In a data communication system, information is transmitted between a sender and a receiver using a character-based approach. To ensure data integrity and synchronization, character stuffing and destuffing techniques are employed. Character stuffing involves adding special control characters before and after a designated delimiter character in the data stream. This helps distinguish between data characters and control characters, ensuring that the receiver can accurately interpret the transmitted information. Destuffing is the process of removing the added control characters during reception.

Character Stuffing:

In character stuffing, the sender follows these rules:

- A predefined delimiter character is chosen (e.g., the character sequence DLE STX).
- Before transmitting the data, the sender inserts a control character (e.g., DLE) before any occurrence of the delimiter character within the data.
- The sender also adds an additional control character (e.g., DLE) at the end of the data.
- The modified data is transmitted to the receiver.

Character Destuffing:

The receiver follows these rules to destuff the received data:

- Upon receiving data, the receiver scans for occurrences of the control character before the delimiter character (e.g., DLE) and removes the control character, treating the delimiter character as a regular data character.
- When the receiver encounters the control character at the end of the data, it simply removes the control character.
- The receiver processes the destuffed data stream, interpreting the original data and recognizing the delimiter character as the end of a data frame.

Problem Scenario:

Imagine two functions are communicating using characterstuffing and characterdestuffing. The sender has a string of characters that need to be transmitted, and the receiver needs to correctly interpret this data.

Input Data (characterstuffing):

This is the data DLE STX that needs DLE to be stuffed DLE ETX with characters.

Output Data (after stuffing and before transmission):

This is the data DLE DLE STX that needs DLE DLE to be stuffed DLE DLE ETX with characters.

Received Data- (characterdestuffing, after transmission and before destuffing):

This is the data DLE DLE STX that needs DLE DLE to be stuffed DLE DLE ETX with characters.

Output Data (after destuffing):

This is the data DLE STX that needs DLE to be stuffed DLE ETX with characters.

Description (Problem Statement)

Task-5.2: Bit Stuffing and Destuffing

In a serial data transmission system, data is sent as a sequence of bits. However, the sender and receiver might not be perfectly synchronized due to differences in clock speeds, signal propagation delays, and other factors. If the sender sends a long sequence of identical bits (e.g., several consecutive 1s or 0s), the receiver might lose track of where the bit boundaries are, leading to synchronization errors and misinterpretation of the data.

Bit stuffing is a technique used to prevent this problem. When transmitting data, the sender inserts an extra "0" bit after a certain number of consecutive identical bits in the data stream. The receiver knows to remove the extra "0" bits and interpret the original data correctly.

The rules for bit stuffing are as follows:

Whenever the sender encounters a sequence of "n" consecutive identical bits (where "n" is a predetermined threshold, typically 5), it inserts an extra "0" bit after the "n-1" consecutive bits.

The receiver recognizes this pattern and removes the extra "0" bit whenever it detects "n" consecutive identical bits followed by a "0" bit.

Problem Scenario:

Imagine two functions are communicating using BitStuffing and BitDestuffing. The sender has a string of bits that need to be transmitted, and the receiver needs to correctly interpret this data.

1. BitStuffing

Input Frame (before bit stuffing):

0111110 1011111101011110111111101 01111110

Output Frame (after bit stuffing):

0111110 101111101010111110011111101101 0111110

2. BitDestuffing

Input Frame (after bit stuffing):

0111110 101111101010111110011111101101 0111110

Output Frame (after destuffing):

0111110 1011111101011110111111101 01111110