**5. WAP to Implement Framing Protocol: Byte Stuffing**

**Steps to Implement Byte Stuffing**

**1. Understand the Concept**

* **Special Characters**:
  + **Start-of-frame (SOF)**: Indicates the beginning of a frame (e.g., '@').
  + **Escape (ESC)**: Used to indicate that the next character is part of the payload, not a control character (e.g., '#').
* **Sender**:
  + Adds SOF at the beginning of the frame.
  + Replaces each SOF and ESC in the payload with an ESC followed by the special character.
* **Receiver**:
  + Reads the frame and removes the ESC before special characters in the payload.

**2. Define the Key Functions**

* **Sender**: Adds SOF and performs byte stuffing.
* **Receiver**: Detects SOF, interprets ESC sequences, and reconstructs the original message.

**3. Code Implementation**

Below is an example program in C:

#include <stdio.h>

#include <string.h>

#define SOF '@' // Start-of-frame marker

#define ESC '#' // Escape character

// Function to perform byte stuffing at the sender's side

void sender(const char \*message, char \*stuffed\_frame) {

int j = 0;

stuffed\_frame[j++] = SOF; // Add SOF at the start of the frame

for (int i = 0; message[i] != '\0'; i++) {

if (message[i] == SOF || message[i] == ESC) {

stuffed\_frame[j++] = ESC; // Add escape character

}

stuffed\_frame[j++] = message[i]; // Add the actual character

}

stuffed\_frame[j++] = SOF; // Add SOF at the end of the frame

stuffed\_frame[j] = '\0'; // Null-terminate the stuffed frame

}

// Function to perform byte unstuffing at the receiver's side

void receiver(const char \*stuffed\_frame, char \*original\_message) {

int j = 0;

for (int i = 1; stuffed\_frame[i] != SOF; i++) { // Skip the initial SOF

if (stuffed\_frame[i] == ESC) {

i++; // Skip the escape character

}

original\_message[j++] = stuffed\_frame[i];

}

original\_message[j] = '\0'; // Null-terminate the original message

}

int main() {

const char \*message = "Hello @World# Protocol";

char stuffed\_frame[100], original\_message[100];

// Step 1: Perform byte stuffing

sender(message, stuffed\_frame);

printf("Stuffed Frame: %s\n", stuffed\_frame);

// Step 2: Perform byte unstuffing

receiver(stuffed\_frame, original\_message);

printf("Original Message: %s\n", original\_message);

return 0;

}

**4. Explanation of the Code**

1. **Sender Function**:
   * Adds an SOF at the start and end of the frame.
   * Scans the message for SOF and ESC.
   * Adds an ESC before any SOF or ESC found in the message.
   * Constructs the stuffed frame.
2. **Receiver Function**:
   * Skips the initial SOF.
   * Detects ESC and skips it before adding the next character.
   * Stops processing upon encountering the final SOF.
3. **Main Function**:
   * Defines a test message.
   * Calls the sender and receiver functions.
   * Displays the stuffed frame and the reconstructed message.

**5. Compilation and Execution**

* Compile the program:

bash

Copy code

gcc -o byte\_stuffing byte\_stuffing.c

* Run the program:

bash

Copy code

./byte\_stuffing

**6. Expected Output**

plaintext

Copy code

Stuffed Frame: @Hello #@World## Protocol@

Original Message: Hello @World# Protocol

**7. Notes**

* The SOF and ESC characters are predefined in the code. You can customize them as needed.
* Ensure the receiver processes the frame correctly by interpreting the ESC character.
* The program simulates communication between a sender and receiver for simplicity.