EX.NO: 03	IMPLEMENTATION OF BIT STUFFING AND CHARACTER STUFFING
DATE:	

AIM:

To implement bit stuffing and character stuffing.

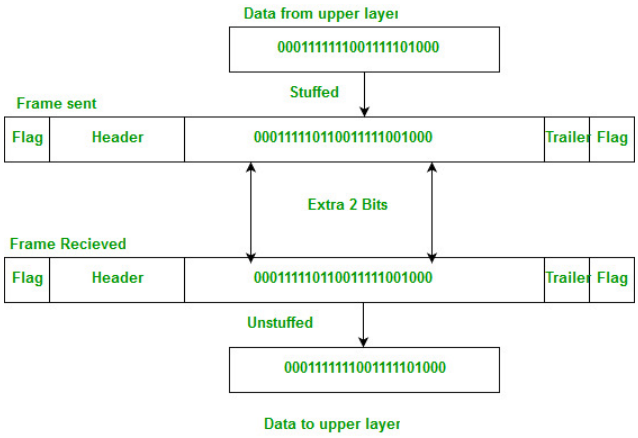
SOFTWARE REQUIRED:

Java online complier

THEORY:

- **BIT STUFFING:**

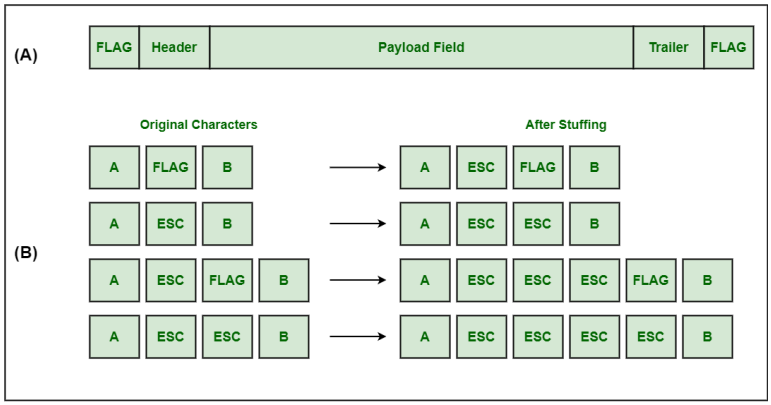
In data transmission and telecommunication, bit stuffing (also known—uncommonly—as positive justification) is the insertion of non-information bits into data. Stuffed bits should not be confused with overhead bits. Bit stuffing is used for various purposes, such as for bringing bit streams that do not necessarily have the same or rationally related bit rates up to a common rate, or to fill buffers or frames. The location of the stuffing bits is communicated to the receiving end of the data link, where these extra bits are removed to return the bit streams to their original bit rates or form. Bit stuffing may be used to synchronize several channels before multiplexing or to rate-match two single channels to each other.



- **CHARACTER STUFFING:**

Character stuffing is a technique used in computer programming to control data transmission between different systems or devices. It involves adding special characters or sequences of characters to the data being transmitted to mark the beginning and end of a data frame.

Character stuffing is commonly used in data communication protocols to ensure the receiving system correctly interprets the transmitted data. It helps frame the data so that the receiver can easily identify the start and end of each data frame. One common use case of character stuffing is serial communication, where data is transmitted one bit at a time over a communication channel.



A Character Stuffing
(A) A frame delimited by flag bytes
(B) Four examples of byte sequences before and after byte stuffing

IMPLEMENTATION CODE FOR BIT STUFFING:

```
import java.util.*;
public class BitStuffing {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the data stream: ");
        String dataStream = sc.nextLine();
        String stuffedStream = bitStuffing(dataStream);
        System.out.println("Original Data Stream: " + dataStream);
        System.out.println("Stuffed Data Stream: " + stuffedStream);
        sc.close();
    }
    public static String bitStuffing(String dataStream) {
        StringBuilder stuffedStream = new StringBuilder();
        int consecutiveOnes = 0;
        for (int i = 0; i < dataStream.length(); i++) {
            char bit = dataStream.charAt(i);
            stuffedStream.append(bit);
            consecutiveOnes = (bit == '1') ? consecutiveOnes + 1 : 0;
            if (consecutiveOnes == 5) {
                stuffedStream.append('0');
                consecutiveOnes = 0;
            } }
        return stuffedStream.toString();
    } }
```

IMPLEMENTATION CODE FOR CHARACTER STUFFING:

```
import java.util.*;
class Char {
    public static void main(String r[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of characters: ");
        int n = sc.nextInt();
        String in[] = new String[n];
        System.out.println("Enter characters: ");
        for (int i = 0; i < n; i++) {
            in[i] = sc.next();
        }
        for (int i = 0; i < n; i++) {
            if (in[i].equals("dle")) {
                in[i] = "dle dle";
            }
        }
        System.out.println("Transmitted message is: ");
        System.out.print(" dle stx ");
        for (int i = 0; i < n; i++) {
            System.out.print(in[i] + " ");
        }
        System.out.println(" dle etx ");
    }
}
```

OUTPUT:

- **BIT STUFFING:**

```
Enter the data stream:
101010111111010010
Original Data Stream: 101010111111010010
Stuffed Data Stream: 1010101111101010010
```

- **CHARACTER STUFFING:**

```
Enter number of characters:
5
Enter characters:
a b c dle e
Transmitted message is:
 dle stx a b c dle dle e  dle etx
```

RESULT:

Thus, the implementation of both bit and character stuffing has been successful.