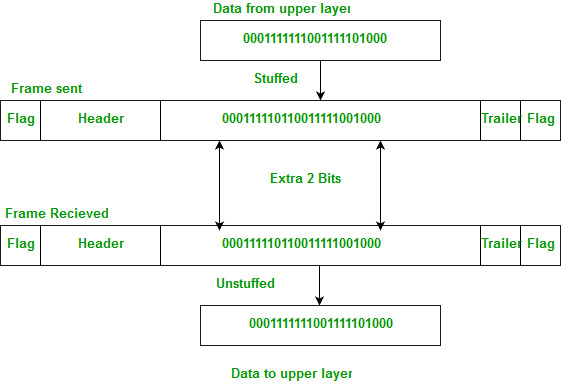
**BIT STUFFING**

The data link layer is responsible for something called Framing, which is the division of stream of bits from network layer into manageable units (called frames). Frames could be of fixed size or variable size. In variable-size framing, we need a way to define the end of the frame and the beginning of the next frame.

**Bit stuffing** is the insertion of non information bits into data. Note that stuffed bits should not be confused with overhead bits.

**Overhead bits** are non-data bits that are necessary for transmission (usually as part of headers, checksums etc.).



**Applications of Bit Stuffing –**

1. synchronize several channels before multiplexing
2. rate-match two single channels to each other
3. run length limited coding

**Run length limited coding –** To limit the number of consecutive bits of the same value(i.e., binary value) in the data to be transmitted. A bit of the opposite value is inserted after the maximum allowed number of consecutive bits.

Bit stuffing technique does not ensure that the sent data is intact at the receiver side (i.e., not corrupted by transmission errors). It is merely a way to ensure that the transmission starts and ends at the correct places.

**Disadvantages of Bit Stuffing:**

* **The code rate is unpredictable; it depends on the data being transmitted.**
* **The stuffed bits do not contain any information.**

**Example of bit stuffing –**   
Bit sequence: 110101111101011111101011111110 (without bit stuffing)   
Bit sequence: 1101011111**0**01011111**0**101011111**0**110 (with bit stuffing)

After 5 consecutive 1-bits, a 0-bit is stuffed. Stuffed bits are marked bold.

#include <stdio.h>

#include <string.h>

void bitStuffing(int N, int arr[])

{

// Here we will store the stuffed array.

int brr[30];

int i, j, k;

i = 0;

j = 0;

while (i < N) {

if (arr[i] == 1) {

//Will store the count of consecutive ones

int count = 1;

brr[j] = arr[i];

for (k = i + 1;

arr[k] == 1 && k < N && count < 5; k++) {

j++;

brr[j] = arr[k];

count++;

// if we found 5 consecutive ones then will insert a zero.

if (count == 5) {

j++;

brr[j] = 0;

}

i = k;

}

}

// otherwise will copy the array directly without any modification.

else {

brr[j] = arr[i];

}

i++;

j++;

}

for (i = 0; i < j; i++)

printf("%d", brr[i]);

}

int main()

{

int N = 9;

int arr[] = { 1, 1, 1, 0, 1, 1, 1, 1, 1 };

bitStuffing(N, arr);

return 0;

}

**Output**

1110111110

**Explanation**

**Explanation of the above code for Bit Stuffing Program in C**  
In the above program we are traversing the original data array and after that when we encounter a set bit we will count the number of consecutive set bits if the theory is 5 then we will insert a zero bit at their end and then repeat the operation for the remaining array.

**Time Complexity:** O(n) will be the time complexity for writing bit stuffing code in C.

**Space Complexity:** O(N) will be the space complexity for bit stuffing program in C Language.

CRC

#include <stdio.h>

#include <string.h>

#define GENERATOR "1001" // CRC-4 (4-bit generator polynomial)

void generateCRC(char \*message, char \*generator, char \*crc) {

int msg\_len = strlen(message);

int gen\_len = strlen(generator);

int crc\_len = gen\_len - 1;

strcpy(crc, message); // Copy message to crc initially

// Perform modulo-2 division

for (int i = 0; i <= msg\_len - crc\_len; i++) {

if (crc[i] == '1') {

for (int j = 0; j < gen\_len; j++) {

crc[i + j] = crc[i + j] == generator[j] ? '0' : '1';

}

}

}

}

int verify CRC(char \*message, char \*generator, char \*received\_crc) {

char crc[100];

generateCRC(message, generator, crc);

// Check if the generated CRC matches the received CRC

if (strcmp (crc, received\_crc) == 0) {

return 1; // No error detected

} else {

return 0; // Error detected

}

}

int main() {

char message[100];

char received\_crc[10];

printf("Enter the message: ");

scanf("%s", message);

printf("Enter the received CRC: ");

scanf("%s", received\_crc);

int is ErrorFree = verify CRC(message, GENERATOR, received\_crc);

if (isErrorFree) {

printf ("Received message is error-free.\n");

} else {

Printf ("Error detected in the received message.\n");

}

return 0;

}