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-D10

Lab Experiment Name: Implementation of Stuffing-DeStuffing & IPV4.

Student Details

	Name	ID
1.	Masum Hossain	221902164

Lab Date	**: 04/03/2024**
Submission Date	**: 25/03/2024**
Course Teacher's Name	**: Rusmita Halim Chaity**

Lab Report Status
Marks: ………………………………… **Signature:.....................**
Comments:... **Date:.............................**

1. TITLE OF THE LAB REPORT EXPERIMENT

Implementation of Stuffing and De-stuffing and IPV4 Implementation.

2. OBJECTIVES

This lab report aims at the implementation of stuffing and de-stuffing and IPV4.
This lab mainly focuses on :

- To acquire knowledge about stuffing and de-stuffing as well as IPv4.
- To implement a bits stuffing and de-stuffing algorithms in C language.
- To develop a program to prompt user input for an IP address and provide options for conversion to binary or vice versa.

3. ANALYSIS

Bit stuffing and de-stuffing ensure data integrity by adding/removing extra bits in communication protocols.Here stuffing involves adding extra bits to data streams for transmission integrity, while de-stuffing removes these added bits upon reception.IPv4 conversion transforms addresses into binary for efficient processing and decimal for readability, aiding network management. Together, these techniques optimize data transmission and enhance network functionality and reliability.

Bit Stuffing & De-Stuffing Algorithm:

For Stuffing:

```
1. Initializing variables for counting consecutive 1s and output index.
2. Copying the header "01111110" into the output buffer.
3. Iterating through the input sequence:
        If the current bit is '1', increment the count and copy it to the output.
        If the current bit is '0', reset the count and copy it to the output.
        If the count reaches 5, insert a '0' and reset the count.
4. Copying the trailer "01111110" to the end of the stuffed output.
```

For De-stuffing:

```
1. Initializing variables for counting consecutive 1s and output index.
2. Iterating through the stuffed input sequence, excluding the header and trailer:
        If the current bit is '1', increment the count and copy it to the output.
        If the current bit is '0', reset the count and copy it to the output.
        If the count reaches 5 and the next bit is '0', skip the next bit and reset the count.
3. Terminating the output buffer with a null character.
```

IP Address Conversion Algorithm:

IPv4 to Binary:

1. Splitting the IPv4 address into octets using strtok.
2. Convert each octet to binary and store it in an array.
3. Printing the binary representation, separating each octet with a space.

Binary to IPv4:

1. Iterating through the binary string in groups of 8 bits.
2. Converting each group to decimal and concatenate it to the IPv4 address string.
3. Printing the resulting IPv4 address.

These algorithms ensure efficient handling of data sequences and IP address conversions, enhancing data transmission and representation capabilities.

4. IMPLEMENTATION

In this section, we critically analyze the implementation of the Bit Stuffing and De-Stuffing Algorithm, implemented in C, ensures data integrity with dedicated functions for both operations, contained by the "01111110" header and trailer. Meanwhile, the IP Address Conversion Program, also coded in C, provides users with functions to classify IP addresses and convert between binary and decimal forms, accessible through a user-friendly menu interface by switch case.

Here is the implementation part of each section to perform bits stuffing and de-stuffing as well as IPv4.

Code for Stuffing and De-stuffing:

```
#include <stdio.h>
#include <string.h>
#define MAX_SIZE 100
void bitStuff(char *input, char *output) {
    int count = 0;
    int outIndex = 0;
    strcpy(output, "01111110");
    outIndex += 8;
    for (int i = 0; i < strlen(input); i++) {
        if (input[i] == '1') {
            count++;
            output[outIndex++] = '1';
        } else {
            count = 0;
            output[outIndex++] = '0';
        if (count == 5) {
            output[outIndex++] = '0';
            count = 0;
        }
    }
    strcpy(output + outIndex, "01111110");
}
void bitDestuff(char *input, char *output) {
    int count = 0;
    int outIndex = 0;

    for (int i = 8; i < strlen(input) - 8; i++) {
        if (input[i] == '1') {
            count++;
            output[outIndex++] = '1';
        } else {
            count = 0;
            output[outIndex++] = '0';
        }

        if (count == 5 && input[i + 1] == '0') {
            count = 0;
            i++;
        }
    }
    output[outIndex] = '\0';
}
int main() {
    char input[MAX_SIZE];
    char stuffed[MAX_SIZE * 2];
    char destuffed[MAX_SIZE];

    printf(" Please enter the Sequence:");
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = '\0';

    bitStuff(input, stuffed);
    printf("Stuffed Output: %s\n", stuffed);

    bitDestuff(stuffed, destuffed);
    printf("De-stuffed Output: %s\n", destuffed);
    return 0;}
```

Code for IPv4:

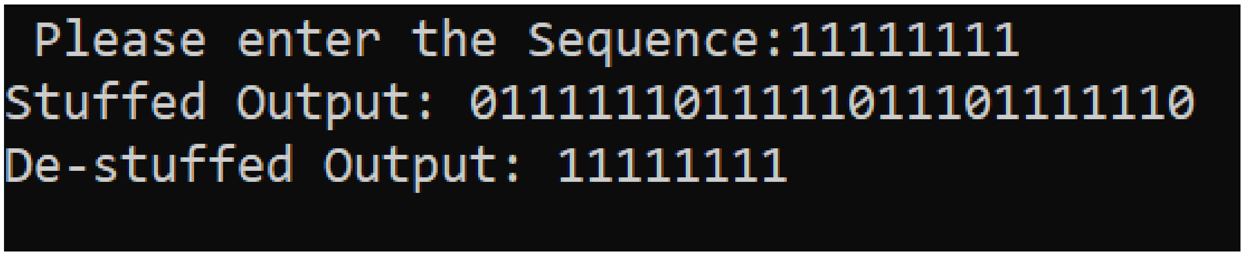
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void ipv4_to_binary(char *ipv4_address) {
   char *token;
   int binary[32] = {0};
   int i = 0;

   token = strtok(ipv4_address, ".");
   while (token != NULL) {
      int octet = atoi(token);
      for (int j = 7; j >= 0; j--) {
         binary[i++] = (octet >> j) & 1;
      }
      token = strtok(NULL, ".");
   }

   for (int k = 0; k < 32; k++) {
      printf("%d", binary[k]);
      if ((k + 1) % 8 == 0) {
         printf(" ");
      }
   }
   printf("\n");
}

void binary_to_ipv4(char *binary_str) {
   char ipv4_address[16] = "";
   int octet = 0;
   for (int i = 0; i < 32; i++) {
      octet = (octet << 1) | (binary_str[i] - '0');
      if ((i + 1) % 8 == 0) {
         sprintf(ipv4_address + strlen(ipv4_address), "%d", octet);
         if (i != 31) {
            strcat(ipv4_address, ".");
         }
         octet = 0;
      }
   }
   printf("IPv4 Address: %s\n", ipv4_address);
}
```

```
int main() {
    char ipv4_address[16];
    char binary_str[33];
    int option;

    printf("Enter an IPv4 address (e.g., '192.168.1.1'): ");
    fgets(ipv4_address, sizeof(ipv4_address), stdin);
    ipv4_address[strcspn(ipv4_address, "\n")] = '\0';

    printf("Choose an option:\n");
    printf("1. Convert IPv4 to Binary\n");
    printf("2. Convert Binary to IPv4\n");
    scanf("%d", &option);

    switch (option) {
        case 1:
            ipv4_to_binary(ipv4_address);
            break;
        case 2:
            printf("Enter a binary string (32 bits): ");
            scanf("%32s", binary_str);
            binary_to_ipv4(binary_str);
            break;
        default:
            printf("Invalid option\n");
            break;
    }

    return 0;}
```

Here,we have implemented both stuffing and de-stuffing by taking user input and defining head and tail is used for the actual data integrity.For IPv4 implementation we have taken user input as it is class A,B or C and converted Decimal to binary and vice versa**.**

5. OUTPUT

The implementation of various test cases has been done for both stuffing and de-stuffing .
 Below are the summarized results of the tests conducted:

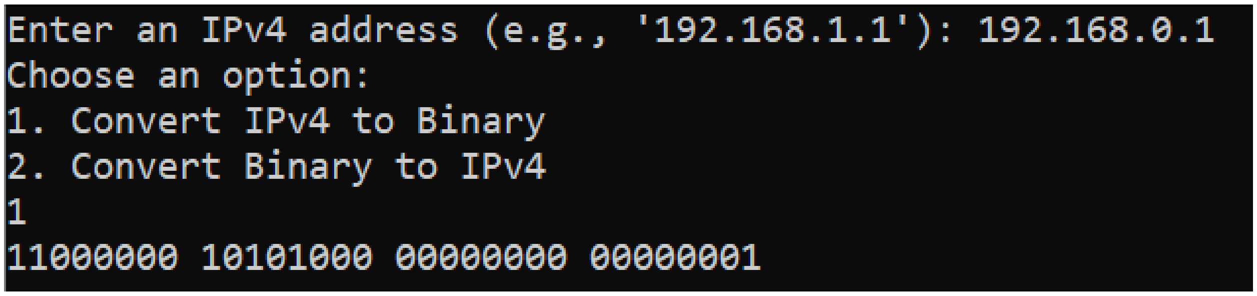


```
 Please enter the Sequence:11111111
Stuffed Output: 01111110111110111011111110
De-stuffed Output: 11111111
```

Fig 01: Showing the output for Stuffing and De-stuffing Bits.

We entered the bit sequence as “11111111” as five consecutive 1 has been occurred 0 will be added in case of stuffing and we have got our desired output.

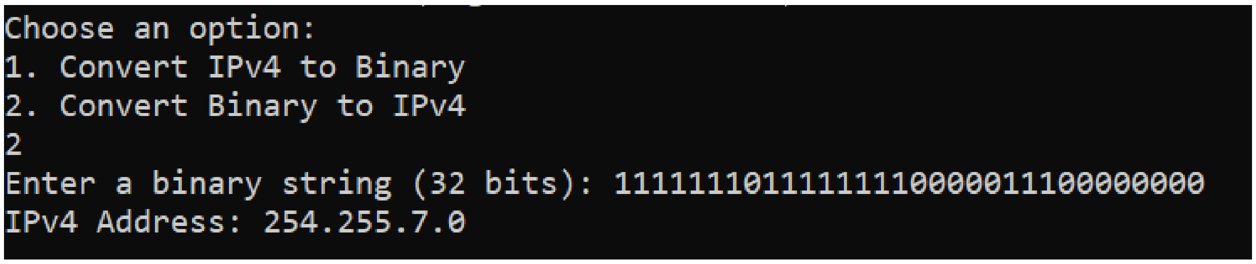
Output for IPv4:



```
Enter an IPv4 address (e.g., '192.168.1.1'): 192.168.0.1
Choose an option:
1. Convert IPv4 to Binary
2. Convert Binary to IPv4
1
11000000 10101000 00000000 00000001
```

Fig 02: Showing the output for IPv4 Decimal to Binary.

Here,we have entered as class C for decimal 192.168.0.1 and converted into binary by switch case input 1.



```
Choose an option:
1. Convert IPv4 to Binary
2. Convert Binary to IPv4
2
Enter a binary string (32 bits): 11111110111111110000011100000000
IPv4 Address: 254.255.7.0
```

Fig 03: Showing the output for IPv4 Binary to Decimal.

Here,we have entered 32 bits as input must contain an octet part in four portions corresponding to binary value to decimal.

Overall, the tests confirmed the successful implementation of Encoding and decoding IPv4 for both decimal to binary and vice versa.All the test cases passed successfully and desired results were obtained.

6. ANALYSIS AND DISCUSSION

The analysis section of the lab report succinctly summarizes the successful implementation and evaluation of both programmed worked and implemented successfully.It's noted that both the bit stuffing and de-stuffing algorithm and the IP address conversion program exhibit linear time complexities of O(n), where 'n' represents the length of the input data. Additionally, the space complexity for both algorithms remains constant, with O(1) space required for the IP address conversion program, and O(n) space required for the bit stuffing and de-stuffing algorithm due to potential duplication of the input sequence. These complexities underscore the algorithms' efficient performance and scalability, ensuring reliable data manipulation and IP address management while minimizing resource consumption.

7. SUMMARY

In summary, both algorithms demonstrate efficient performance with linear time complexities and minimal space requirements, ensuring reliable data handling and IP address management and successful implementation of both programs.