**Vishwakarma Institute of Technology, Pune**

**Department of Information Technology & MCA**

**Academic Year: 2020-2021**

DCAN COURSE PROJECT

Year: SY Batch: 3

Team Members:

|  |  |  |  |
| --- | --- | --- | --- |
| Sr.No. | Name of Project Group Members | Roll No. | GR No. |
| 1 | Akash Sivanandan | 5 | 11910346 |
| 2 | Prapti Maheshwari | 37 | 11910360 |
| 3 | Tanishka Shetty | 63 | 11910175 |
| 4 | Vaibhavi Shetty | 73 | 11910971 |
| 5 | Neeraj Vazalwar | 75 | 11911361 |

Topic: Character Stuffing and Destuffing

Abstract:

The data link layer is the protocol layer in a program that handles the moving of data into and out of a physical link in a network. The data link layer is Layer 2 in the Open Systems Interconnection ([OSI](https://searchnetworking.techtarget.com/definition/OSI)) architecture model for a set of telecommunication protocols. Data bits are encoded, decoded and organized in the data link layer, before they are transported as frames between two adjacent nodes on the same [LAN](https://searchnetworking.techtarget.com/definition/local-area-network-LAN) or [WAN](https://searchnetworking.techtarget.com/definition/WAN-wide-area-network).

The data link layer takes the packets it gets from the network layer and encapsulates them into frames for transmission. Each frame contains a frame header, a payload field for holding the packet, and a frame trailer, as illustrated in Fig. 1.



Fig. 1

Framing is a point-to-point connection between two computers or devices consists of a wire in which data is transmitted as a stream of bits. Frames have headers that contain information such as error-checking codes.

Character Stuffing and Destuffing framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes. In the past, the starting and ending bytes were different, but in recent years most protocols have used the same byte, called a flag byte, as both the starting and ending delimiter, as shown in Fig. 2 as FLAG. In this way, if the receiver ever loses synchronization, it can just search for the flag byte to find the end of the current frame. Two consecutive flag bytes indicate the end of one frame and start of the next one.

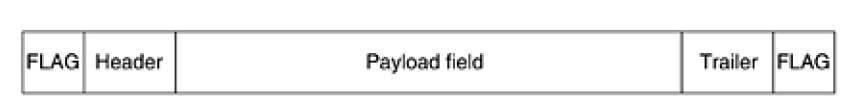


Fig. 2

The data link layer on the receiving end removes the escape byte before the data are given to the network layer. This technique is called byte stuffing or character stuffing. Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it.

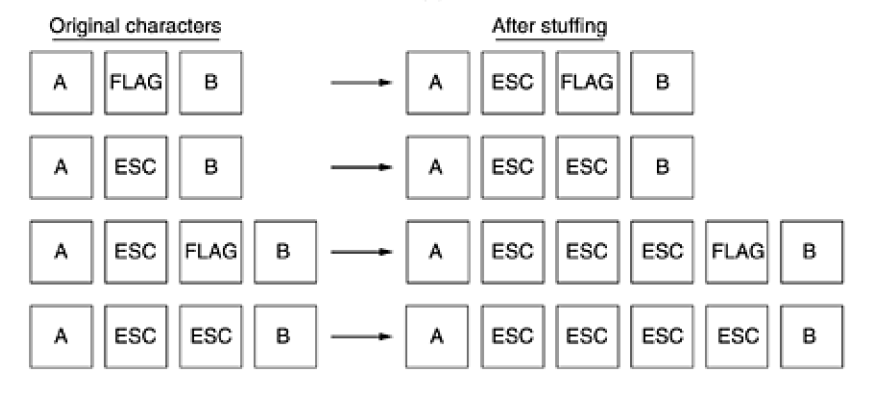


Fig. 3

**Code:**

SERVER:

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<ctype.h>

#define PORT 4455

int asciiValueToBinary(int asciiInput)

{

int res = 0, i = 1, rem;

while (asciiInput > 0)

{

rem = asciiInput % 2;

res = res + (i \* rem);

asciiInput = asciiInput / 2;

i = i \* 10;

}

return(res);

}

int main()

{

char d[50],s[50];

int i=0,j=1;

printf("\nEnter the codeword : ");

scanf("%s",d);

s[0]='$'; //Here $ is used as a flag .It must be at start and end of a frame.

while(d[i]!='\0')

{

if((d[i]=='$')||(d[i]=='@'))

{

s[j]='@';

j++;

}

s[j]=d[i];

i++,j++;

}

s[j]='$',j++,s[j ]='\0';

printf("\nStuffed frame(data sent) : %s\n",s);

printf("\n\n\t Character \t ASCII \t\t\t Binary\n");

int x=0;

while (s[x]!='\0')

{

printf("\t %c \t\t %d \t\t\t %d \n", s[x], toascii(s[x]), asciiValueToBinary(toascii(s[x])));

x++;

}

printf("\n\nSent data in Binary format (Frame)");

x=0;

while (s[x]!='\0')

{

printf("%d", asciiValueToBinary(toascii(s[x])));

x++;

}

printf("\n");

int sockfd;

struct sockaddr\_in serverAddr;

int newSocket;

struct sockaddr\_in newAddr;

socklen\_t addr\_size;

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

memset(&serverAddr, '\0', sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET; //

serverAddr.sin\_port = htons(PORT);

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

bind(sockfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

listen(sockfd, 5);

addr\_size = sizeof(newAddr);

newSocket = accept(sockfd,( struct sockaddr\*)&newAddr, &addr\_size);

send(newSocket, s, strlen(s),0);

printf("\nLength of sent data is %ld\n",strlen(s));

return 0;

}

CLIENT (receiver):

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<ctype.h>

#define PORT 4455

int asciiValueToBinary(int asciiInput);

int main()

{

int clientSocket;

struct sockaddr\_in serverAddr;

char buffer[1024];

clientSocket = socket(AF\_INET, SOCK\_STREAM,0); //socket create

memset(&serverAddr, '\0', sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET; //

serverAddr.sin\_port = htons(PORT);

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

connect(clientSocket, (struct sockaddr\*)&serverAddr,sizeof(serverAddr));

int len,k;

printf("\nEnter length of sent data: ");

scanf("%d",&len);

recv(clientSocket, buffer, len, 0);

// printf("\nData Received: %s",buffer);

char d[50],s[50];

for (k=0;k<len;k++)

{

strcpy(d,buffer);

}

d[k]='\0';

int i=1,j=0;

// printf("\nEnter the code word from sender side :"); //Include the flag($) in that code word.

// scanf("%s",d);

printf("\nFrame received: ");

int x=0;

while (d[x]!='\0')

{

printf("%d", asciiValueToBinary(toascii(d[x])));

x++;

}

printf("\n");

printf("\n\n\t Character \t ASCII \t\t\t Binary\n");

x=0;

while (d[x]!='\0')

{

printf("\t %c \t\t %d \t\t\t %d \n", d[x], toascii(d[x]), asciiValueToBinary(toascii(d[x])));

x++;

}

printf("\nData Received: %s",d);

while(d[i+1]!='\0')

{

if(d[i]=='@'&&(d[i+1]=='@'||d[i+1]=='$'))

{

i++;

}

s[j]=d[i];

i++;

j++;

}

s[j]='\0';

printf("\n\nOriginal code word is : %s\n",s);

return 0;

}

int asciiValueToBinary(int asciiInput)

{

int res = 0, i = 1, rem;

while (asciiInput > 0)

{

rem = asciiInput % 2;

res = res + (i \* rem);

asciiInput = asciiInput / 2;

i = i \* 10;

}

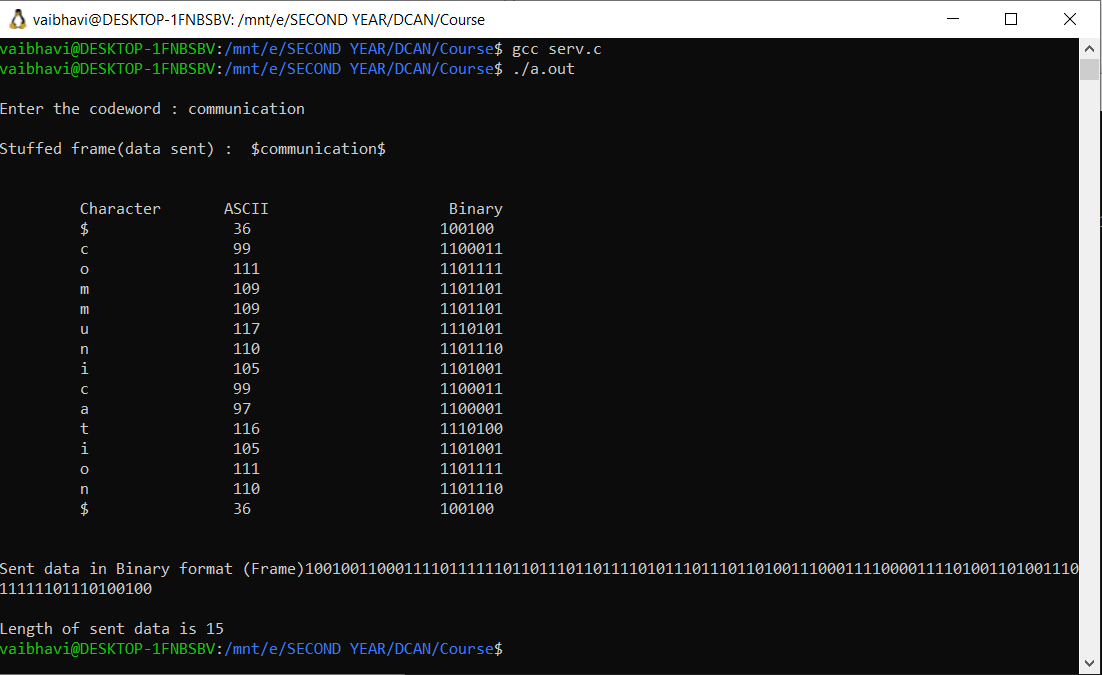
return(res);

}

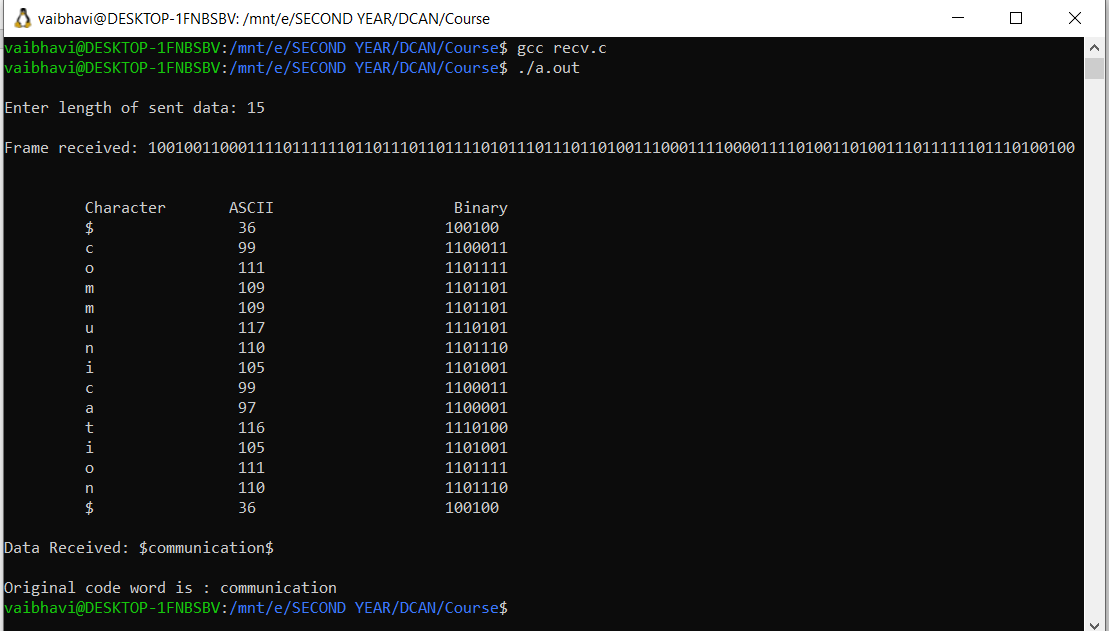
**OUTPUT:**

CASE 1:

Server:

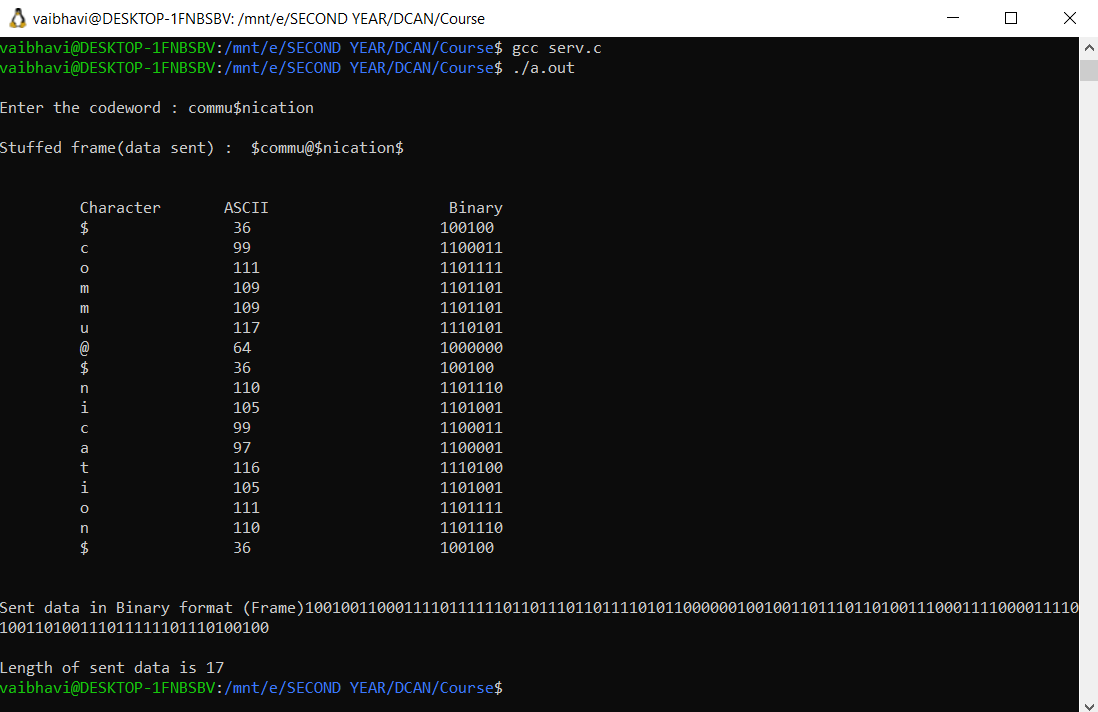


Client:

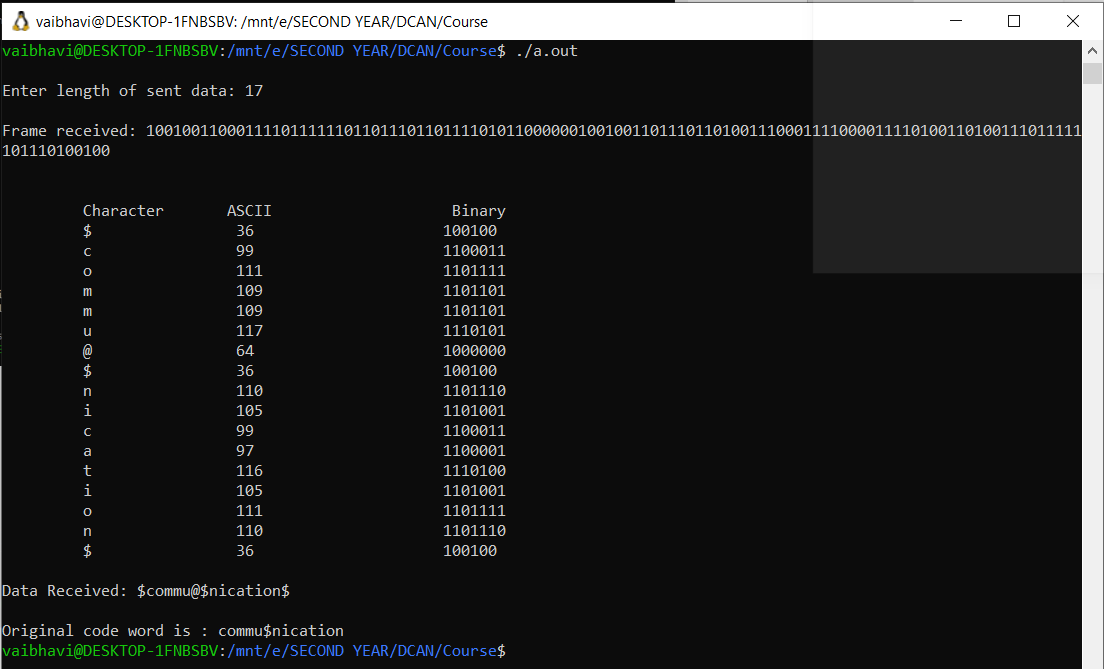


CASE 2:

Server:

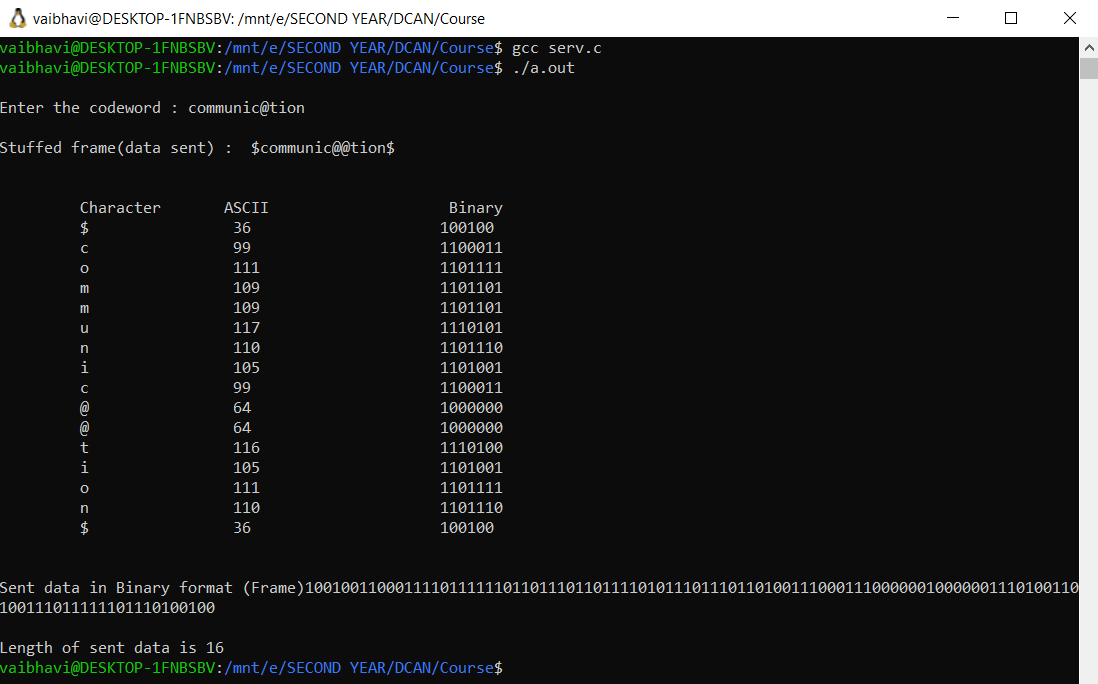


Client:

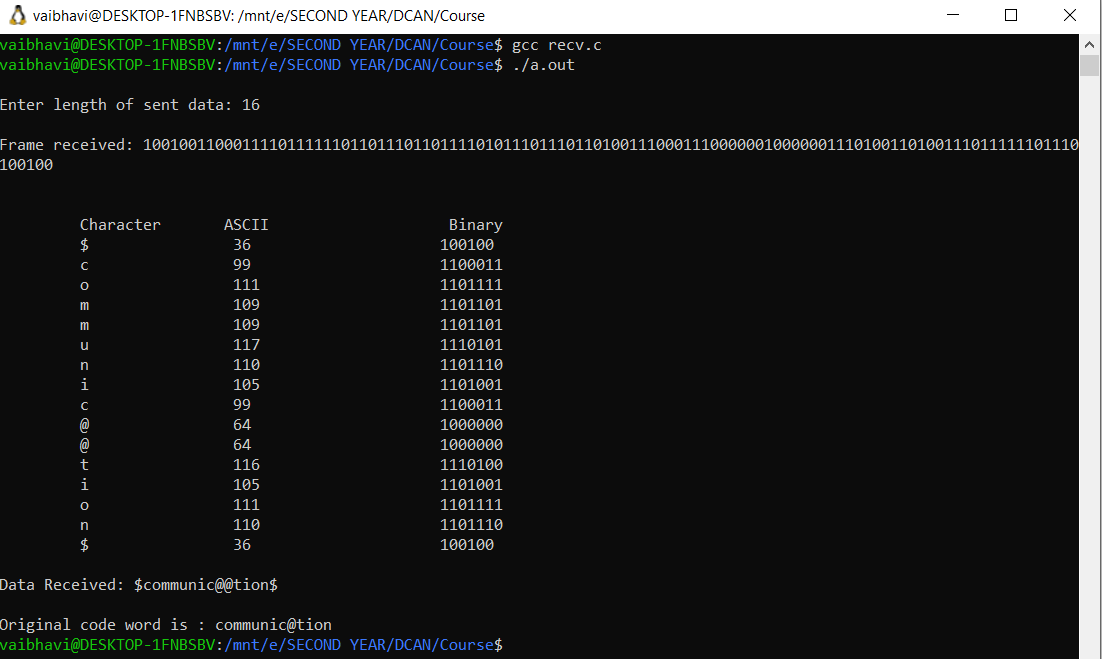


CASE 3:

Server:

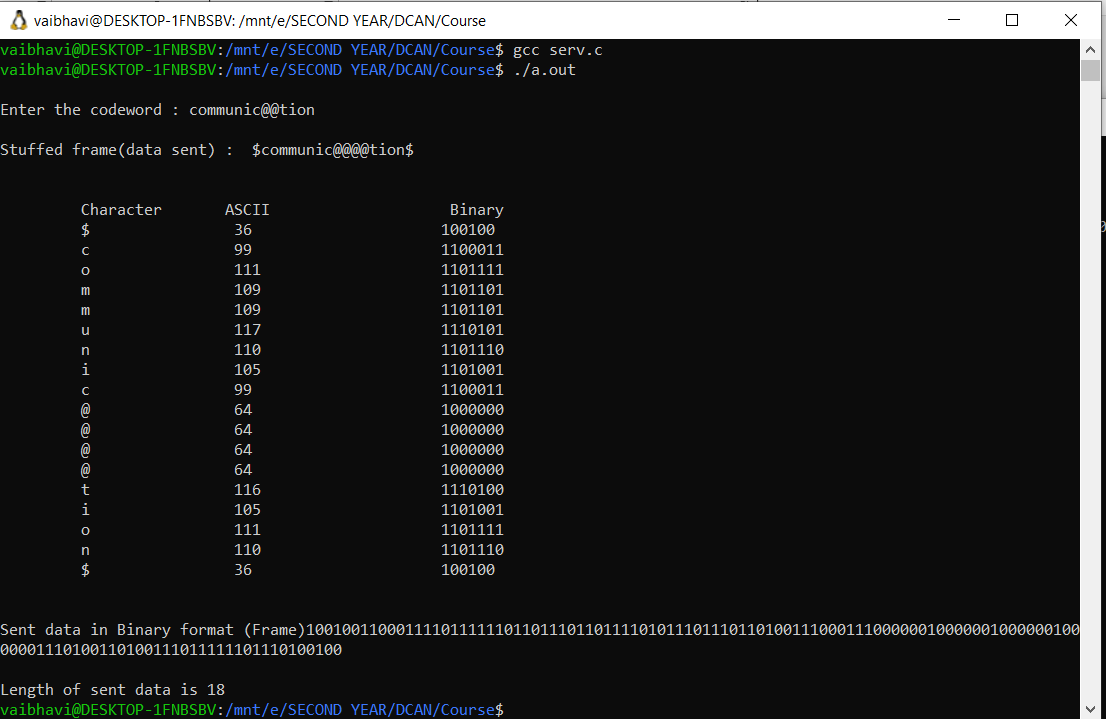


Client:

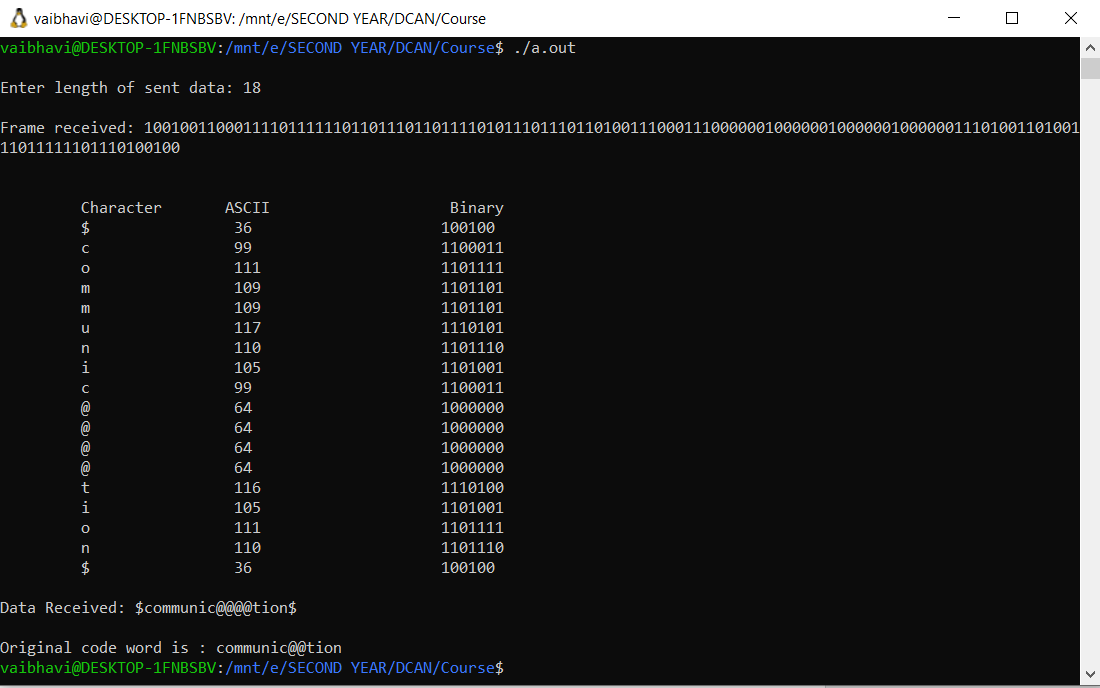


CASE 4:

Server:

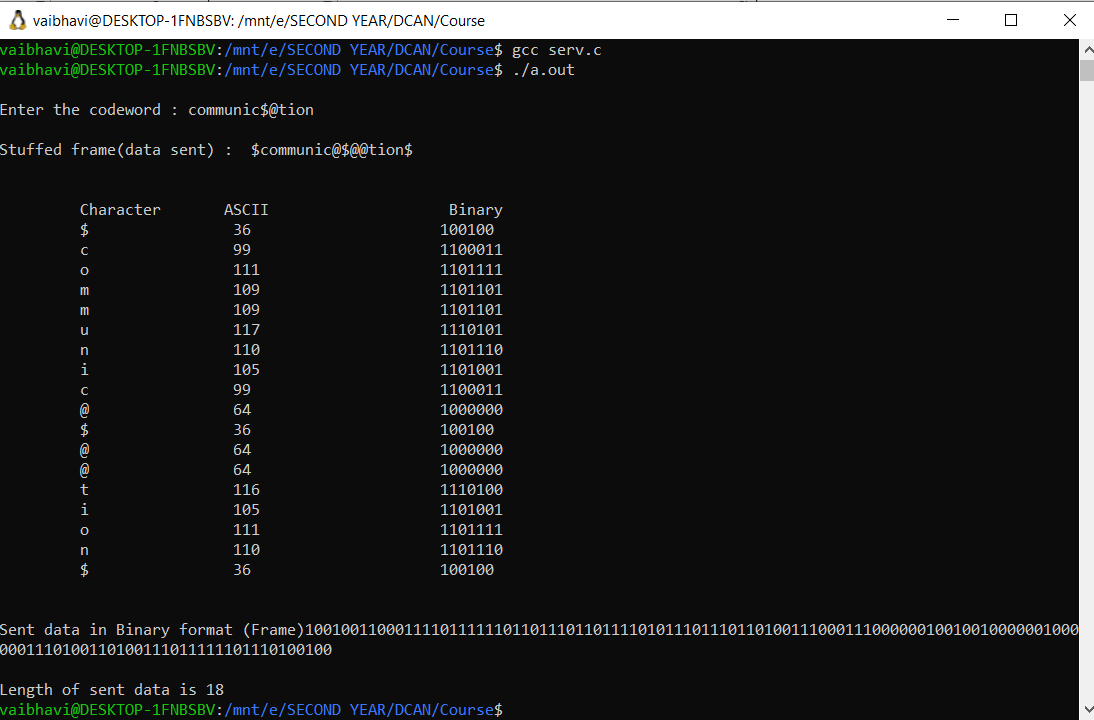


Client:



CASE 5:

Server:



Client:

