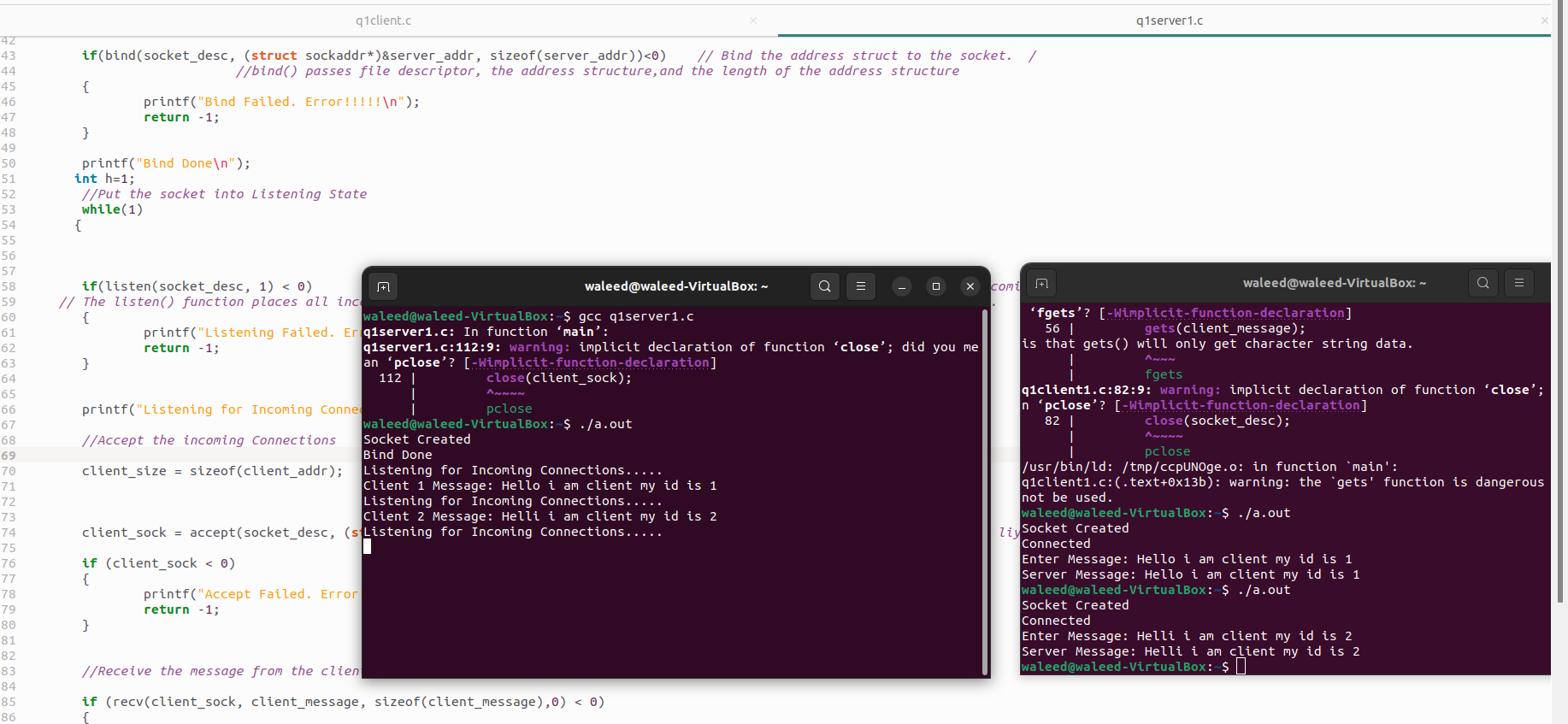
**COMPUTER NETWORKS**

**LAB 5**

**QUESTION1**



**Question 1 code:**

**Client side**

#include <stdio.h>  
#include <string.h>  
#include <sys/socket.h> //socket  
#include <arpa/inet.h> //inet\_addr  
  
int main(void)  
{  
        int socket\_desc;  
        struct sockaddr\_in server\_addr;  
        char server\_message[2000], client\_message[2000];  
         
        //Cleaning the Buffers  
         
        memset(server\_message,'\0',sizeof(server\_message));  
        memset(client\_message,'\0',sizeof(client\_message));  
         
        //Creating Socket  
         
        socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);  
         
        if(socket\_desc < 0)  
        {  
                printf("Could Not Create Socket. Error!!!!!\n");  
                return -1;  
        }  
         
        printf("Socket Created\n");  
         
        //Specifying the IP and Port of the server to connect  
         
        server\_addr.sin\_family = AF\_INET;  
        server\_addr.sin\_port = htons(2000);  
        server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  
         
        //Now connecting to the server accept() using connect() from client side  
         
        if(connect(socket\_desc, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0)  
        {  
                printf("Connection Failed. Error!!!!!");  
                return -1;  
        }  
         
        printf("Connected\n");  
         
        //Get Input from the User  
         
        printf("Enter Message: ");  
        gets(client\_message);                                     //One is that gets() will only get character string data.  
                                                       // will get only one variable at a time.  
//  reads characters from stdin and loads them into str  
        //Send the message to Server  
         
        if(send(socket\_desc, client\_message, strlen(client\_message),0) < 0)  
        {  
                printf("Send Failed. Error!!!!\n");  
                return -1;  
        }  
         
        //Receive the message back from the server  
         
        if(recv(socket\_desc, server\_message, sizeof(server\_message),0) < 0)  
        {  
                printf("Receive Failed. Error!!!!!\n");  
                return -1;  
        }  
         
        printf("Server Message: %s\n",server\_message);  
         
        memset(server\_message,'\0',sizeof(server\_message));  
        memset(client\_message,'\0',sizeof(client\_message));

        //Closing the Socket

close(client\_sock);  
        close(socket\_desc);  
        return 0;        
}

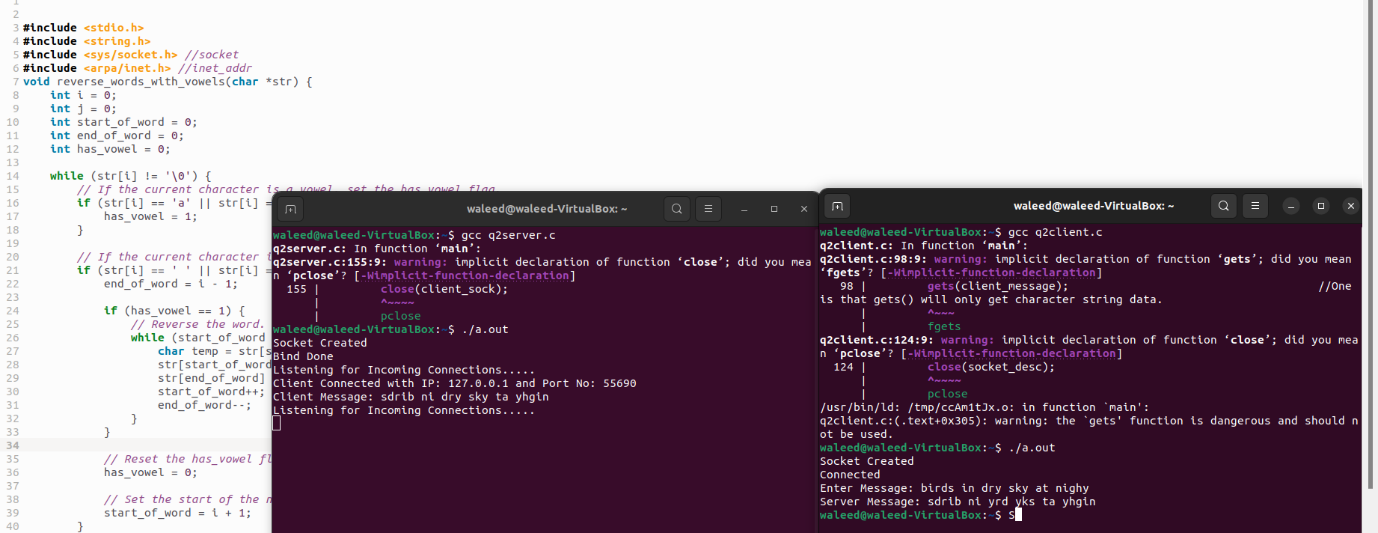
        close(socket\_desc);  
         
        return 0;  
}

**Server Side**

#include <stdio.h>  
#include <string.h>  
#include <sys/socket.h> //socket  
#include <arpa/inet.h> //inet\_addr  
  
int main(void)  
{  
         
        int socket\_desc, client\_sock, client\_size;  
        struct sockaddr\_in server\_addr, client\_addr;         //SERVER ADDR will have all the server address  
        char server\_message[2000], client\_message[2000];                 // Sending values from the server and receive from the server we need this  
  
        //Cleaning the Buffers  
         
        memset(server\_message,'\0',sizeof(server\_message));  
        memset(client\_message,'\0',sizeof(client\_message));     // Set all bits of the padding field//  
         
        //Creating Socket  
         
        socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);  
         
        if(socket\_desc < 0)  
        {  
                printf("Could Not Create Socket. Error!!!!!\n");  
                return -1;  
        }  
         
        printf("Socket Created\n");  
         
        //Binding IP and Port to socket  
         
        server\_addr.sin\_family = AF\_INET;               /\* Address family = Internet \*/  
        server\_addr.sin\_port = htons(2000);               // Set port number, using htons function to use proper byte order \*/  
        server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");    /\* Set IP address to localhost \*/  
  
  
  
// BINDING FUNCTION  
         
        if(bind(socket\_desc, (struct sockaddr\*)&server\_addr, sizeof(server\_addr))<0)    // Bind the address struct to the socket.  /  
                            //bind() passes file descriptor, the address structure,and the length of the address structure  
        {  
                printf("Bind Failed. Error!!!!!\n");  
                return -1;  
        }          
         
        printf("Bind Done\n");  
       int h=1;  
        //Put the socket into Listening State  
        while(1)  
       {  
         
         
         
        if(listen(socket\_desc, 1) < 0)                               //This listen() call tells the socket to listen to the incoming connections.  
     // The listen() function places all incoming connection into a "backlog queue" until accept() call accepts the connection.  
        {  
                printf("Listening Failed. Error!!!!!\n");  
                return -1;  
        }  
         
       
        printf("Listening for Incoming Connections.....\n");  
         
        //Accept the incoming Connections  
         
        client\_size = sizeof(client\_addr);  
  
  
  
        client\_sock = accept(socket\_desc, (struct sockaddr\*)&client\_addr, &client\_size);          // heree particular client k liye new socket create kr rhaa ha  
         
        if (client\_sock < 0)  
        {  
                printf("Accept Failed. Error!!!!!!\n");  
                return -1;  
        }  
         
  
        //Receive the message from the client  
         
        if (recv(client\_sock, client\_message, sizeof(client\_message),0) < 0)  
        {  
                printf("Receive Failed. Error!!!!!\n");  
                return -1;  
        }  
         
        printf("Client %d ", h);  
        printf("Message: %s\n",client\_message);  
        h++;  
        //Send the message back to client  
         
        strcpy(server\_message, client\_message);  
         
        if (send(client\_sock, server\_message, strlen(client\_message),0)<0)  
        {  
                printf("Send Failed. Error!!!!!\n");  
                return -1;  
        }  
         
        memset(server\_message,'\0',sizeof(server\_message));  
        memset(client\_message,'\0',sizeof(client\_message));  
         
        }  
         
           
        //Closing the Socket  
         
        close(client\_sock);  
        close(socket\_desc);  
        return 0;        
}

        close(socket\_desc);  
         
        return 0;  
}

**QUESTION 2**



**Server code:**

/\*

        TCP\_Server. This Program will will create the Server side for TCP\_Socket Programming.

        It will receive the data from the client and then send the same data back to client.

\*/

#include <stdio.h>

#include <string.h>

#include <sys/socket.h> //socket

#include <arpa/inet.h> //inet\_addr

void reverse\_words\_with\_vowels(char \*str) {

    int i = 0;

    int j = 0;

    int start\_of\_word = 0;

    int end\_of\_word = 0;

    int has\_vowel = 0;

    while (str[i] != '\0') {

        // If the current character is a vowel, set the has\_vowel flag.

        if (str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u') {

            has\_vowel = 1;

        }

        // If the current character is a space or the end of the string, reverse the word if it has at least one vowel.

        if (str[i] == ' ' || str[i] == '\0') {

            end\_of\_word = i - 1;

            if (has\_vowel == 1) {

                // Reverse the word.

                while (start\_of\_word < end\_of\_word) {

                    char temp = str[start\_of\_word];

                    str[start\_of\_word] = str[end\_of\_word];

                    str[end\_of\_word] = temp;

                    start\_of\_word++;

                    end\_of\_word--;

                }

            }

            // Reset the has\_vowel flag.

            has\_vowel = 0;

            // Set the start of the next word.

            start\_of\_word = i + 1;

        }

        i++;

    }

if (has\_vowel == 0 && start\_of\_word < i) {

end\_of\_word = i - 1;

while (start\_of\_word < end\_of\_word) {

char temp = str[start\_of\_word];

str[start\_of\_word] = str[end\_of\_word];

str[end\_of\_word] = temp;

start\_of\_word++;

end\_of\_word--;

}

}

}

int main(void)

{

        int socket\_desc, client\_sock, client\_size;

        struct sockaddr\_in server\_addr, client\_addr;         //SERVER ADDR will have all the server address

        char server\_message[2000], client\_message[2000];                 // Sending values from the server and receive from the server we need this

        //Cleaning the Buffers

        memset(server\_message,'\0',sizeof(server\_message));

        memset(client\_message,'\0',sizeof(client\_message));     // Set all bits of the padding field//

        //Creating Socket

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

        if(socket\_desc < 0)

        {

                printf("Could Not Create Socket. Error!!!!!\n");

                return -1;

        }

        printf("Socket Created\n");

        //Binding IP and Port to socket

        server\_addr.sin\_family = AF\_INET;               /\* Address family = Internet \*/

        server\_addr.sin\_port = htons(2000);               // Set port number, using htons function to use proper byte order \*/

        server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");    /\* Set IP address to localhost \*/

        // BINDING FUNCTION

        if(bind(socket\_desc, (struct sockaddr\*)&server\_addr, sizeof(server\_addr))<0)    // Bind the address struct to the socket.  /

                                    //bind() passes file descriptor, the address structure,and the length of the address structure

        {

                printf("Bind Failed. Error!!!!!\n");

                return -1;

        }

        printf("Bind Done\n");

        //Put the socket into Listening State

        while (1){

        if(listen(socket\_desc, 1) < 0)                               //This listen() call tells the socket to listen to the incoming connections.

     // The listen() function places all incoming connection into a "backlog queue" until accept() call accepts the connection.

        {

                printf("Listening Failed. Error!!!!!\n");

                return -1;

        }

        printf("Listening for Incoming Connections.....\n");

        //Accept the incoming Connections

        client\_size = sizeof(client\_addr);

        client\_sock = accept(socket\_desc, (struct sockaddr\*)&client\_addr, &client\_size);          // heree particular client k liye new socket create kr rhaa ha

        if (client\_sock < 0)

        {

                printf("Accept Failed. Error!!!!!!\n");

                return -1;

        }

        printf("Client Connected with IP: %s and Port No: %i\n",inet\_ntoa(client\_addr.sin\_addr),ntohs(client\_addr.sin\_port));

               //inet\_ntoa() function converts the Internet host address in, given in network byte order, to a string in IPv4 dotted-decimal notation

        //Receive the message from the client

        if (recv(client\_sock, client\_message, sizeof(client\_message),0) < 0)

        {

                printf("Receive Failed. Error!!!!!\n");

                return -1;

        }

        reverse\_words\_with\_vowels(client\_message);

        printf("Client Message: %s\n",client\_message);

        //Send the message back to client

        strcpy(server\_message, client\_message);

        if (send(client\_sock, server\_message, strlen(client\_message),0)<0)

        {

                printf("Send Failed. Error!!!!!\n");

                return -1;

        }

        memset(server\_message,'\0',sizeof(server\_message));

        memset(client\_message,'\0',sizeof(client\_message));

        }

        //Closing the Socket

        close(client\_sock);

        close(socket\_desc);

        return 0;

}

**Client code:**

/\*

        TCP\_Client. This Program will implement the Client Side for TCP\_Socket Programming.

        It will get some data from user and will send to the server and as a reply from the

        server, it will get its data back.

\*/

#include <stdio.h>

#include <string.h>

#include <sys/socket.h> //socket

#include <arpa/inet.h> //inet\_addr

void reverse\_words\_without\_vowels(char \*str) {

    int i = 0;

    int j = 0;

    int start\_of\_word = 0;

    int end\_of\_word = 0;

    int has\_vowel = 0;

    while (str[i] != '\0') {

        // If the current character is a vowel, set the has\_vowel flag.

        if (str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u') {

            has\_vowel = 1;

        }

        // If the current character is a space or the end of the string, reverse the word if it has no vowel characters.

        if (str[i] == ' ' || str[i] == '\0') {

            end\_of\_word = i - 1;

            if (has\_vowel == 0) {

                // Reverse the word.

                while (start\_of\_word < end\_of\_word) {

                    char temp = str[start\_of\_word];

                    str[start\_of\_word] = str[end\_of\_word];

                    str[end\_of\_word] = temp;

                    start\_of\_word++;

                    end\_of\_word--;

                }

            }

            // Reset the has\_vowel flag.

            has\_vowel = 0;

            // Set the start of the next word.

            start\_of\_word = i + 1;

        }

        i++;

    }

if (has\_vowel == 0 && start\_of\_word < i) {

end\_of\_word = i - 1;

while (start\_of\_word < end\_of\_word) {

char temp = str[start\_of\_word];

str[start\_of\_word] = str[end\_of\_word];

str[end\_of\_word] = temp;

start\_of\_word++;

end\_of\_word--;

}

}

}

int main(void)

{

        int socket\_desc;

        struct sockaddr\_in server\_addr;

        char server\_message[2000], client\_message[2000];

        //Cleaning the Buffers

        memset(server\_message,'\0',sizeof(server\_message));

        memset(client\_message,'\0',sizeof(client\_message));

        //Creating Socket

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

        if(socket\_desc < 0)

        {

                printf("Could Not Create Socket. Error!!!!!\n");

                return -1;

        }

        printf("Socket Created\n");

        //Specifying the IP and Port of the server to connect

        server\_addr.sin\_family = AF\_INET;

        server\_addr.sin\_port = htons(2000);

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

        //Now connecting to the server accept() using connect() from client side

        if(connect(socket\_desc, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0)

        {

                printf("Connection Failed. Error!!!!!");

                return -1;

        }

        printf("Connected\n");

        //Get Input from the User

        printf("Enter Message: ");

        gets(client\_message);                                     //One is that gets() will only get character string data.

                                                       //       will get only one variable at a time.

                                                                //  reads characters from stdin and loads them into str

        //Send the message to Server

        if(send(socket\_desc, client\_message, strlen(client\_message),0) < 0)

        {

                printf("Send Failed. Error!!!!\n");

                return -1;

        }

        //Receive the message back from the server

        if(recv(socket\_desc, server\_message, sizeof(server\_message),0) < 0)

        {

                printf("Receive Failed. Error!!!!!\n");

                return -1;

        }

        reverse\_words\_without\_vowels(server\_message);

        printf("Server Message: %s\n",server\_message);

        memset(server\_message,'\0',sizeof(server\_message));

        memset(client\_message,'\0',sizeof(client\_message));

        //Closing the Socket

        close(socket\_desc);

        return 0;

}