Q. Write a socket program to implement TCP client and Server such that server should be able to send text to client and client check that the received number of words in the sentence.

Server code:

```
// Server

#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#include <iostream>
#include <string> // for string

using namespace std;

#define PORT 8080
#define REQUEST_QUEUE_LEN 3
#define NUMBER_BASE 10

int calculateWord(const char *sentence)
{
 int words = 0;
```

```
 int i = 0;
 while (sentence[i] != '\0')
 {
  if (sentence[i] == ' ')
   words++;
  i++;
 }
 words++;
 return words;
}

int intermediateFunc(char message[])
{
 char *strPtr = NULL;
 int result = strtol(message, &strPtr, NUMBER_BASE);
 return result;
}

class Server
{
private:
 struct sockaddr_in address;
 int server_fd;
 int numberOfClients;
 static char buffer[1024];
 static void sendMsg(int, int, const char *);
 static void closeConnection(int, const char *);
 static bool recvMsg(int);
 static void sendToAll(const char *);
 static void recvSendStructure(int, bool);

public:
 Server(int);
 void operator()(bool);
 ~Server();
};
char Server::buffer[1024] = {};

Server::Server(int port = PORT)
{
 srand(time(0));

 this->numberOfClients = 0;
```

```
int opt = 1;

// Creating socket file descriptor
server_fd = socket(AF_INET, SOCK_STREAM, 0);
if (server_fd < 0)
{
 perror("Socket creation failed");
 exit(EXIT_FAILURE);
}

// Forcefully attaching socket to the port 8080
if (setsockopt(server_fd, SOL_SOCKET,
       SO_REUSEADDR | SO_REUSEPORT, &opt,
       sizeof(opt)))
{
 perror("setsockopt");
 exit(EXIT_FAILURE);
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// Forcefully attaching socket to the port 8080
if (bind(server_fd, (struct sockaddr *)&address,
    sizeof(address)) < 0)
{
 perror("bind failed");
 exit(EXIT_FAILURE);
}
if (listen(server_fd, REQUEST_QUEUE_LEN) < 0)
{
 perror("listen");
 exit(EXIT_FAILURE);
}

 cout << "Server started ... \n";
}

Server::~Server()
{
 shutdown(server_fd, SHUT_RDWR);
 cout << "Server shutdown.\n";
}
```

```
void Server::closeConnection(int new_socket, const char *message = "Closed connection,
new_socket(%d)\n")
{
 close(new_socket);
 printf(message, new_socket);
}

bool Server::recvMsg(int new_socket)
{
 int valread = read(new_socket, buffer, sizeof(buffer));
 if (strcmp(buffer, "!exit") == 0)
 {
  closeConnection(new_socket);
  return false;
 }

 if (valread ≤ 0)
  return false;

 buffer[valread] = '\0';
 return true;
}

void Server::sendMsg(int from_socket, int to_socket, const char *message = "\0")
{
 if (*message ≠ '\0')
  send(to_socket, message, strlen(message), 0);
}

void Server::recvSendStructure(int new_socket, bool alwaysReceive = false)
{
 do
 {
  cout << "Enter sentence to be sent : ";
  scanf("%[^\n]%*c", buffer);
  int numberOfWords = calculateWord(buffer);
  sendMsg(new_socket, new_socket, buffer);
  if (!recvMsg(new_socket))
   break;

  printf("server  > %s\n", buffer);
  printf("expected message > %d\n", numberOfWords);
  printf("client(%d) > %s\n", new_socket, buffer);
  if (numberOfWords == intermediateFunc(buffer))
```

```
    sendMsg(new_socket, new_socket, "Accepted\n");
   else
    sendMsg(new_socket, new_socket, "Rejected\n");

  } while (alwaysReceive);

  closeConnection(new_socket);
}

void Server::operator()(bool alwaysReceive = false)
{
  int addrlen = sizeof(address);

  cout << "Server listning ... \n";
  while (true)
  {
   int new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen);
   if (new_socket < 0)
   {
    perror("accept");
    exit(EXIT_FAILURE);
   }
   cout << "New conection made, new_soc ? : " << new_socket << endl;
   numberOfClients++;
   recvSendStructure(new_socket, alwaysReceive);
  }
}

int main(int argc, char const *argv[])
{
  Server server;
  server();
  return 0;
}
```

Client code:

```
// Client

#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#include <iostream>
#include <sstream>  // for string streams
#include <stdlib.h> // for exit()

#define PORT 8080
#define NUMBER_BASE 10

using namespace std;

bool IS_CONNECTED = true;

char buffer[1024] = {0};

/**
* Calculates number of non alphabetic character
*/
int calculateWord(char *sentence)
{
 int words = 0;
 int i = 0;
 while (sentence[i] != '\0')
 {
  if (sentence[i] == ' ')
   words++;
  i++;
 }
 words++;
 return words;
}

/**
```

```
* Calculates number of words and then converts (int) to (char *)
*/
string intermediateFunc(char message[])
{
 int words = calculateWord(message);
 ostringstream str1;
 str1 << words;
 string temp = str1.str();
 strcpy(message, temp.c_str());
 return temp;
}

void connect(sockaddr_in &serv_addr, int &sock, int &client_fd)
{
 /**
  * int sock = socket(DOMAIN, TYPE, PROTOCOL)
  * DOMAIN: integer, specifies communication domain.
  * type: communication type
    SOCK_STREAM: TCP(reliable, connection oriented)
    SOCK_DGRAM: UDP(unreliable, connectionless)
  * Create a new socket of type TYPE in domain DOMAIN, using
    protocol PROTOCOL.  If PROTOCOL is zero, one is chosen automatically.
    Returns a file descriptor for the new socket, or -1 for errors.
  */
 if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
 {
  printf("\n Socket creation error \n");
  exit(1);
 }

 serv_addr.sin_family = AF_INET;
 serv_addr.sin_port = htons(PORT);

 // Convert IPv4 and IPv6 addresses from text to binary form
 if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) ≤ 0)
 {
  printf("\nInvalid address/ Address not supported \n");
  exit(1);
 }

 if ((client_fd = connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr))) < 0)
 {
  printf("\nConnection Failed \n");
  exit(1);
```

```
 }
}

void sendMsg(int sock, int client_fd)
{
 int key = 3;

 if (!IS_CONNECTED)
  exit(1);

 if (strcmp(buffer, "!exit") == 0)
 {
  // closing the connected socket
  send(sock, buffer, strlen(buffer), 0);
  close(client_fd);
  printf("Connection closed\n");
  exit(1);
 }
 intermediateFunc(buffer);
 cout << "Number of words : " << buffer << endl;
 send(sock, buffer, strlen(buffer), 0);
}

void recvMsg(int sock, int client_fd)
{
 int valread;
 valread = read(sock, buffer, 1024);
 if (valread ≤ 0)
 {
  // closing the connected socket
  close(client_fd);
  printf("Server down. Closed connection, client_fd(%d)\n", client_fd);
  IS_CONNECTED = false;
  exit(EXIT_FAILURE);
  exit(1);
 }
 buffer[valread] = '\0';
 printf("Received message : %s\n", buffer);
}

int main(int argc, char const *argv[])
{
 struct sockaddr_in serv_addr;
```

```
/**
 * sock: socket descriptor, an integer (like a file-handle)
 */
int sock = 0;

int client_fd = -1;

connect(serv_addr, sock, client_fd);
IS_CONNECTED = true;

recvMsg(sock, client_fd);
sendMsg(sock, client_fd);
recvMsg(sock, client_fd);

close(client_fd);
printf("Connection closed\n");
return 0;
}
```

OUTPUT :

Server:

```
> ./server
Server started...
Server listning...
New conection made, new_soc ? : 4
Enter sentence to be sent : Great ideas often receive violent opposition from mediocre minds.
server  > 9
expected message > 9
client(4) > 9
Closed connection, new_socket(4)
New conection made, new_soc ? : 4
Enter sentence to be sent : Sometimes the biggest act of courage is a small one.
server  >  0
expected message >  0
client(4) >  0
Closed connection, new_socket(4)
```

Client:

```
> ./client
Received message : Great ideas often receive violent opposition from mediocre minds.
Number of words : 9
Received message : Accepted

Connection closed
```

```
> ./client
Received message : Sometimes the biggest act of courage is a small one.
Number of words : 10
Received message : Accepted

Connection closed
```