```
#include <stdio.h>              // standard input/output functions, printf() function
#include <unistd.h>             // close(socket) function
#include <stdlib.h>             // exit(0) function
#include <sys/socket.h>         // sockaddr structure
#include <arpa/inet.h>          // htons(), htonl() functions, inet_addr structure
#include <string.h>             // memset() function
#define MAX_PENDING 5
#define BUFFER_SIZE 32

int main(){
/*..........................................................................................
    1. Create a server socket using the socket(domain, type, protocol) function call
        int domain = AF_INET       ==> IPv4 communication domain
        int type = SOCK_STREAM     ==> sequenced, reliable, two-way, connection-based byte streams
        int protocol = IPPROTO_TCP ==> TCP
............................................................................................
    int server_socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(server_socket < 0){
        printf("An error occurred while opening a server socket.\n");
        exit(0);
    }
    printf("Server socket successfully created.\n");
/*..........................................................................................
    2. Address format:
        struct sockaddr_in{
            sa_family_t sin_family;   ==> address family: AF_INET
            in_port_t sin_port;       ==> port in network byte order
            struct in_addr sin_addr;  ==> internet address
        };
    Internet address:
        struct in_addr{
            uint32_t s_addr;          ==> address in network byte order
        };
............................................................................................
    struct sockaddr_in serverAddress;
    memset(&serverAddress, 0, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(12345);
    serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
    printf("Server Address assigned.\n");
/*..........................................................................................
    3. Bind the socket with a port address using the bind() call
        int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
        sockfd  ==> socket file descriptor
        *addr   ==> pointer to the address created above
        addrLen ==> size of the addr structure
............................................................................................
    int temp = bind(server_socket, (struct sockaddr*) &serverAddress, sizeof(serverAddress));
    if(temp < 0){
        printf("An error occurred while binding server socket with socket address.\n");
        exit(0);
    }
    printf("Server socket was successfully bound to socket address.\n");
/*..........................................................................................
    4. Listen for connections on the socket using listen() function call
        int listen(int sockfd, int backlog);
        sockfd  ==> socket file descriptor
        backlog ==> maximum length to which the queue of pending connections for sockfd may grow
```

```c
..........................................................................................
    int temp1 = listen(server_socket, MAX_PENDING);
    if(temp1 < 0){
        printf("An error occurred while exwcuting listen.\n");
        exit(0);
    }
    printf("The server socket is now listening.\n");
/*.........................................................................................
    5. Accept connections on socket using the accept() function call
        int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
        sockfd  ==> socket file descriptor
        *addr   ==> pointer to the address created above
        addrLen ==> size of the addr structure
..........................................................................................
    char message[BUFFER_SIZE];
    struct sockaddr_in clientAddress;
    int client_length = sizeof(clientAddress);
    int client_socket = accept(server_socket, (struct sockaddr*)&clientAddress, &client_length);
    if(client_socket < 0){
        printf("An error occurred in the client socket.\n");
        exit(0);
    }
    printf("Handling client socket %s\n", inet_ntoa(clientAddress.sin_addr));
/*.........................................................................................
    6. Receive messages from socket using the recv() function call
        ssize_t recv(int sockfd, void *buf, size_t len, int flags);
        sockfd  ==> socket file descriptor
        buf     ==> pointer to buffer where message will be stored
        len     ==> length of buffer, ie. maximum length of incoming message
        flags   ==> bitwise OR of various flags
..........................................................................................
    int temp2 = recv(client_socket, message, BUFFER_SIZE, 0);
    if(temp2 < 0){
        printf("An error occurred while receiving the message.\n");
        exit(0);
    }
    message[temp2] = '\0';
    //float r_num = atoi(message);  //For the exercise where the client sends a real number and server returns integral ceiling
    printf("%s", message);

    printf("Enter a message for the client...\n");
    fgets(message, BUFFER_SIZE, stdin);
/*.........................................................................................
    7. Send a message on a socket using the send() function call
        ssize_t send(int sockfd, const void *buf, size_t len, int flags);
        sockfd  ==> socket file descriptor
        buf     ==> pointer to buffer where message will be stored
        len     ==> length of buffer, ie. maximum length of incoming message
        flags   ==> bitwise OR of various flags
..........................................................................................
    int bytes_sent = send(client_socket, message, strlen(message), 0);
    if(bytes_sent != strlen(message)){
        printf("An error occurred while sending the message to the client.\n");
        exit(0);
    }
    close(server_socket);
}
```