// LAB1 - Q1

```
// client.c
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define NET_BUF_SIZE 1024
#define PORT 8080
void clearBuf(char* b)
{
        int i;
        for (i = 0; i < NET_BUF_SIZE; i++)
                b[i] = '\0';
}

int main(int argc, char const* argv[])
{
        char port[4];
        char server_ip[16];
        printf("Enter server IP address\n");
    scanf("%s",server_ip);
        printf("Enter server Port Number\n");
        scanf("%s",port);
   int status, valread, client_fd;
   struct sockaddr_in serv_addr;
   char val[NET_BUF_SIZE];
   char buffer[1024] = { 0 };
   if ((client_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
           printf("\n Socket creation error \n");
           return -1;
   }

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

   if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
      printf("\nInvalid address/ Address not supported \n");
      return -1;
   }
   if ((status = connect(client_fd, (struct sockaddr*)&serv_addr, sizeof(serv_addr))) < 0) {
      printf("\nConnection Failed \n");
      return -1;
   }
   while(1){
    clearBuf(val);
    printf("Input Message: \n");
    scanf("%s",val);
    send(client_fd, val, strlen(val), 0);
    valread = read(client_fd, buffer, 1024 - 1);

    int value1,value2;
    char comp1[] = "Goodbye";
    char comp2[] = "OK";
    value1 = strcmp(buffer,comp1);
    value2 = strcmp(buffer,comp2);

    if(value1 == 0){
      printf("%s\n",buffer);
      break;
    }else if(value2 == 0){
      printf("%s\n", buffer);
    }else{
      printf("%s\n", buffer);
      printf("Wrong message send by server\n");
    }
   }
   // closing the connected socket
   close(client_fd);
   return 0;
}

// server.c
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080
#define NET_BUF_SIZE 32

void clearBuf(char* b)
{
        int i;
        for (i = 0; i < 1024; i++)
                b[i] = '\0';
```

```
}

int main(int argc, char const* argv[])
{
    int server_fd, new_socket;
    ssize_t valread;
    struct sockaddr_in address;
    int opt = 1;
    socklen_t addrlen = sizeof(address);
    char buffer[1024] = { 0 };
    char net_buf[NET_BUF_SIZE];

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      perror("socket failed");
      exit(EXIT_FAILURE);
    }

    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);

    if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
      perror("bind failed");
      exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0) {
      perror("listen");
      exit(EXIT_FAILURE);
    }
    if ((new_socket = accept(server_fd, (struct sockaddr*)&address, &addrlen)) < 0) {
      perror("accept");
      exit(EXIT_FAILURE);
    }

    while(1){
      memset(buffer,0, sizeof(buffer));
      valread = read(new_socket, buffer, 1024 - 1); // subtract 1 for the null
      printf("%s\n",buffer);
```

```
   char cmp[] = "Bye";
   int value;
   value = strcmp(cmp,buffer);

   if(value == 0){
     char val[] = "Goodbye";
     send(new_socket, val, strlen(val), 0);

   }else{

   char val[] = "OK";
   send(new_socket, val, strlen(val), 0);
}
  }

  // closing the connected socket
  close(new_socket);
  // closing the listening socket
  close(server_fd);
  return 0;
}
```

// LAB-1 Q-2

```
// client.c
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8000
int main(int argc, char const *argv[])
{
      char port[4];
      char server_ip[16];
      printf("Enter server IP address\n");
   scanf("%s",server_ip);
      printf("Enter server Port Number\n");
      scanf("%s",port);

  int status, valread, client_fd;
  struct sockaddr_in serv_addr;
  char *hello = "Hello from client";
```

```
    char buffer[1024] = {0};
    if ((client_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0)
    {
        printf(
            "\nInvalid address/ Address not supported \n");
        return -1;
    }
    if ((status = connect(client_fd, (struct sockaddr *)&serv_addr,
                          sizeof(serv_addr))) < 0)
    {
        printf("\nConnection Failed \n");
        return -1;
    }
    while (1)
    {
        char userExpression[100];
        scanf("%s", userExpression);
        send(client_fd, userExpression, strlen(userExpression), 0);
        valread = read(client_fd, buffer, 1024 - 1);
            if(buffer != "0.00"){
            printf("%s\n", buffer);
            }
    }
    // closing the connected socket
    close(client_fd);
    return 0;
}

//server.c
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <limits.h>
```

```
#define PORT 8000
double EvalExpression(char *expression)
{
    double num1, num2, result;
    char opp = 'a';
    sscanf(expression, "%lf %c %lf", &num1, &opp, &num2);
    switch (opp)
    {
    case '+':
        result = num1 + num2;
        break;
    case '-':
        result = num1 - num2;
        break;
    case '*':
        result = num1 * num2;
        break;
    case '/':
        // Check for division by zero
        if (num2 != 0)
        {
            result = num1 / num2;
        }
        else
        {
            printf("Error: Division by zero is undefined.\n");
            return -9999.00; // Return 0 in case of division by zero
        }
        break;
    default:
        printf("Error in expression %s.\n",expression);
        return -9999.00; // Return 0 for invalid operators
    }
    return result;
}
int main(int argc, char const *argv[])
{
    int server_fd, new_socket;
    ssize_t valread;
    struct sockaddr_in address;
    int opt = 1;
    socklen_t addrlen = sizeof(address);
    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
```

```
{
   perror("socket failed");
   exit(EXIT_FAILURE);
}
// Forcefully attaching socket to the port 8080
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &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, 3) < 0)
{
   perror("listen");
   exit(EXIT_FAILURE);
}
if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                         &addrlen)) < 0)
{
   perror("accept");
   exit(EXIT_FAILURE);
}

while (1)
{
   char buffer[1024] = {0};
   valread = read(new_socket, buffer, 1024 - 1);
   double result = EvalExpression(buffer);
      if(result == -9999.00){
               memset(buffer,0,sizeof(buffer));
               buffer[0] = 'E';
               buffer[1] = 'R';
               buffer[2] = 'R';


               buffer[3] = '\0';
```

```
                    buffer[4] = '\0';
                    buffer[5] = '\0';
                    buffer[6] = '\0';
        }else{
        snprintf(buffer, sizeof(buffer), "%.2f", result);
        }
      send(new_socket, buffer, strlen(buffer), 0);
                    memset(buffer,0,sizeof(buffer));
   }
   close(new_socket);

   // closing the listening socket
   close(server_fd);
   return 0;
}
```

//LAB-2 Q-1

```
//server.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 12345
#define BUFFER_SIZE 1024

void handle_clients(int client1_socket, int client2_socket) {
   char received_char;
   recv(client1_socket, &received_char, sizeof(received_char), 0);
   printf("Server received character: %c\n", received_char);

   // Decrement the character to the preceding letter in the alphabet
   char decremented_char = received_char - 1;

   send(client2_socket, &decremented_char, sizeof(decremented_char), 0);
   printf("Server sent decremented character to Client2: %c\n", decremented_char);
}

int main() {
   int server_socket, client1_socket, client2_socket;
   struct sockaddr_in server_address, client1_address, client2_address;
```

```
    socklen_t client1_address_len = sizeof(client1_address);
    socklen_t client2_address_len = sizeof(client2_address);

    // Create socket
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Error creating socket");
        exit(EXIT_FAILURE);
    }

    // Bind socket to address
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(PORT);

    if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) == -1) {
        perror("Error binding socket");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_socket, 2) == -1) {
        perror("Error listening for connections");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    printf("Server listening on port %d\n", PORT);

    // Accept connection from Client1
    client1_socket = accept(server_socket, (struct sockaddr *)&client1_address,
&client1_address_len);
    if (client1_socket == -1) {
        perror("Error accepting connection from Client1");
        close(server_socket);
        exit(EXIT_FAILURE);
    }
    printf("Client1 connected\n");

    // Accept connection from Client2
    client2_socket = accept(server_socket, (struct sockaddr *)&client2_address,
&client2_address_len);
    if (client2_socket == -1) {
        perror("Error accepting connection from Client2");
        close(server_socket);
        close(client1_socket);
        exit(EXIT_FAILURE);
    }
    printf("Client2 connected\n");

    // Handle communication between clients
    handle_clients(client1_socket, client2_socket);

    // Close sockets
    close(client1_socket);
    close(client2_socket);
    close(server_socket);

    return 0;
}

//client1.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define SERVER_IP "127.0.0.1"
#define SERVER_PORT 12345

int main() {
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1) {
        perror("Error creating socket");
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(SERVER_PORT);
    if (inet_pton(AF_INET, SERVER_IP, &server_address.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        close(client_socket);
        exit(EXIT_FAILURE);
    }
```

```c
    // Connect to the server
    if (connect(client_socket, (struct sockaddr *)&server_address, sizeof(server_address)) == -1)
{
        perror("Error connecting to server");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    // Send a character to the server
    char character_to_send = 'G';
    send(client_socket, &character_to_send, sizeof(character_to_send), 0);

    // Close socket
    close(client_socket);

    return 0;
}

//client2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define SERVER_IP "127.0.0.1"
#define SERVER_PORT 12345

int main() {
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1) {
        perror("Error creating socket");
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(SERVER_PORT);
    if (inet_pton(AF_INET, SERVER_IP, &server_address.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        close(client_socket);
        exit(EXIT_FAILURE);
    }
    // Connect to the server
    if (connect(client_socket, (struct sockaddr *)&server_address, sizeof(server_address)) == -1)
{
        perror("Error connecting to server");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    // Receive and print the decremented character from the server
    char received_char;
    recv(client_socket, &received_char, sizeof(received_char), 0);
    printf("Client2 received decremented character from server: %c\n", received_char);

    // Close socket
    close(client_socket);

    return 0;
}
```

//LAB-2 Q-2

```
// server.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

struct Part {
    int partNumber;
    char partName[255];
    float partPrice;
    int partQuantity;
    int userAccountNumber;
    char partDescription[1024];
};

struct Part partsDatabase[100] = {
    {1, "Engine Oil", 20.0, 50, 12345, "High-quality engine oil for all vehicles."},
    {2, "Brake Pads", 35.5, 30, 56789, "Durable brake pads for smooth braking."},
    {3, "Air Filter", 10.8, 40, 11111, "Efficient air filter for improved engine performance."},
    // Add more sample data as needed
};
```

```
void searchParts(int clientSocket, char* criteria) {
    char message[1024];
    strcpy(message, "Search results: ");

    for (int i = 0; i < sizeof(partsDatabase) / sizeof(partsDatabase[0]); i++) {
        if (strstr(partsDatabase[i].partName, criteria) != NULL) {
            char partInfo[512];
            snprintf(partInfo, sizeof(partInfo), "Part Number: %d, Name: %s, Price: %.2f, Quantity:
%d\n",
                     partsDatabase[i].partNumber, partsDatabase[i].partName,
                     partsDatabase[i].partPrice, partsDatabase[i].partQuantity);
            strcat(message, partInfo);
        }
    }

    send(clientSocket, message, sizeof(message), 0);
}

void getPartName(int clientSocket, int partNumber) {
    char message[1024];
    for (int i = 0; i < sizeof(partsDatabase) / sizeof(partsDatabase[0]); i++) {
        if (partsDatabase[i].partNumber == partNumber) {
            snprintf(message, sizeof(message), "Part Name for Part Number %d: %s", partNumber,
partsDatabase[i].partName);
            send(clientSocket, message, sizeof(message), 0);
            return;
        }
    }

    strcpy(message, "Part not found");
    send(clientSocket, message, sizeof(message), 0);
}

void checkPartAvailability(int clientSocket, int partNumber) {
    char message[1024];
    for (int i = 0; i < sizeof(partsDatabase) / sizeof(partsDatabase[0]); i++) {
        if (partsDatabase[i].partNumber == partNumber) {
            snprintf(message, sizeof(message), "Available Quantity for Part Number %d: %d",
partNumber, partsDatabase[i].partQuantity);
            send(clientSocket, message, sizeof(message), 0);
            return;
        }
    }
    strcpy(message, "Part not found");
    send(clientSocket, message, sizeof(message), 0);
}

void placeOrder(int clientSocket, int partNumber, int orderQuantity, int userAccountNumber) {
    char message[1024];
    for (int i = 0; i < sizeof(partsDatabase) / sizeof(partsDatabase[0]); i++) {
        if (partsDatabase[i].partNumber == partNumber) {
            if (partsDatabase[i].partQuantity >= orderQuantity) {
                partsDatabase[i].partQuantity -= orderQuantity;
                snprintf(message, sizeof(message), "Order placed successfully for Part Number %d,
Quantity: %d", partNumber, orderQuantity);
            } else {
                snprintf(message, sizeof(message), "Insufficient quantity for Part Number %d",
partNumber);
            }

            send(clientSocket, message, sizeof(message), 0);
            return;
        }
    }

    strcpy(message, "Part not found");
    send(clientSocket, message, sizeof(message), 0);
}

int main() {
    int serverSocket, clientSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t addrLen = sizeof(struct sockaddr);

    // Create socket
    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Setup server address struct
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(8888);  // Use any port you prefer
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    // Bind the socket
    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) == -1) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(serverSocket, 5) == -1) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    printf("Server listening on port 8888...\n");

    // Accept incoming connections
    clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddr, &addrLen);
    if (clientSocket == -1) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    printf("Connection accepted from %s:%d\n", inet_ntoa(clientAddr.sin_addr),
ntohs(clientAddr.sin_port));

    // Handle client requests
    char command[256];
    while (1) {
        // Receive command from the client
        recv(clientSocket, command, sizeof(command), 0);

        // Handle different commands
        if (strcmp(command, "SEARCH") == 0) {
            char criteria[256];
            recv(clientSocket, criteria, sizeof(criteria), 0);
            searchParts(clientSocket, criteria);
        } else if (strcmp(command, "GET_NAME") == 0) {
            int partNumber;
            recv(clientSocket, &partNumber, sizeof(partNumber), 0);
            getPartName(clientSocket, partNumber);
        } else if (strcmp(command, "CHECK_AVAILABILITY") == 0) {
            int partNumber;
            recv(clientSocket, &partNumber, sizeof(partNumber), 0);
            checkPartAvailability(clientSocket, partNumber);
        } else if (strcmp(command, "PLACE_ORDER") == 0) {
            int partNumber, orderQuantity, userAccountNumber;
            recv(clientSocket, &partNumber, sizeof(partNumber), 0);
            recv(clientSocket, &orderQuantity, sizeof(orderQuantity), 0);
            recv(clientSocket, &userAccountNumber, sizeof(userAccountNumber), 0);
            placeOrder(clientSocket, partNumber, orderQuantity, userAccountNumber);
        } else if (strcmp(command, "EXIT") == 0) {
            printf("Client disconnected\n");
            break;
        } else {
            // Handle unknown command
            char message[1024];
            snprintf(message, sizeof(message), "Unknown command: %s", command);
            send(clientSocket, message, sizeof(message), 0);
        }
    }

    close(serverSocket);
    close(clientSocket);
    return 0;
}


//client.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;

    // Create socket
    clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Setup server address struct
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(8888);  // Use the same port as the server
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");  // localhost
    // Connect to the server
    if (connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) == -1) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    printf("Connected to the server\n");

    // Handle user input and send commands to the server
    char command[256];
    while (1) {
        printf("Enter command (SEARCH, GET_NAME, CHECK_AVAILABILITY, PLACE_ORDER,
EXIT): ");
        scanf("%s", command);
        send(clientSocket, command, sizeof(command), 0);

        if (strcmp(command, "SEARCH") == 0) {
            char criteria[256];
            printf("Enter search criteria: ");
            scanf("%s", criteria);
            send(clientSocket, criteria, sizeof(criteria), 0);

            // Receive and print the server's response
            char response[1024];
            recv(clientSocket, response, sizeof(response), 0);
            printf("Server response: %s\n", response);
        } else if (strcmp(command, "GET_NAME") == 0) {
            int partNumber;
            printf("Enter part number: ");
            scanf("%d", &partNumber);
            send(clientSocket, &partNumber, sizeof(partNumber), 0);

            // Receive and print the server's response
            char response[1024];
            recv(clientSocket, response, sizeof(response), 0);
            printf("Server response: %s\n", response);
        } else if (strcmp(command, "CHECK_AVAILABILITY") == 0) {
            int partNumber;
            printf("Enter part number: ");
            scanf("%d", &partNumber);
            send(clientSocket, &partNumber, sizeof(partNumber), 0);

            // Receive and print the server's response
            char response[1024];
            recv(clientSocket, response, sizeof(response), 0);
            printf("Server response: %s\n", response);
        } else if (strcmp(command, "PLACE_ORDER") == 0) {
            int partNumber, orderQuantity, userAccountNumber;
            printf("Enter part number: ");
            scanf("%d", &partNumber);
            printf("Enter order quantity: ");
            scanf("%d", &orderQuantity);
            printf("Enter user account number: ");
            scanf("%d", &userAccountNumber);

            send(clientSocket, &partNumber, sizeof(partNumber), 0);
            send(clientSocket, &orderQuantity, sizeof(orderQuantity), 0);
            send(clientSocket, &userAccountNumber, sizeof(userAccountNumber), 0);

            // Receive and print the server's response
            char response[1024];
            recv(clientSocket, response, sizeof(response), 0);
            printf("Server response: %s\n", response);
        } else if (strcmp(command, "EXIT") == 0) {
            break;
        } else {
            printf("Unknown command. Please try again.\n");
        }
    }

    close(clientSocket);
    return 0;
}
```

// LAB-3 Q-1

```
// Server Code
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>

#define MAX_CLIENTS 10
#define MAX_MESSAGE_LEN 1024
```

```
int clients[MAX_CLIENTS];
int client_count = 0;
pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;

void send_to_all_clients(char *message, int current_client) {
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < client_count; i++) {
        if (clients[i] != current_client) {
            if (send(clients[i], message, strlen(message), 0) < 0) {
                perror("Failed to send message");
                continue;
            }
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}

void *handle_client(void *arg) {
    int client_socket = *((int *)arg);
    char message[MAX_MESSAGE_LEN];

    while (1) {
        int receive_size = recv(client_socket, message, MAX_MESSAGE_LEN, 0);
        if (receive_size <= 0) {
            close(client_socket);

            pthread_mutex_lock(&clients_mutex);
            for (int i = 0; i < client_count; i++) {
                if (clients[i] == client_socket) {
                    for (int j = i; j < client_count - 1; j++) {
                        clients[j] = clients[j + 1];
                    }
                    break;
                }
            }
            client_count--;
            pthread_mutex_unlock(&clients_mutex);

            break;
        }
        message[receive_size] = '\0';
        printf("%s\n",message);
        send_to_all_clients(message, client_socket);
    }
    pthread_exit(NULL);
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    pthread_t client_threads[MAX_CLIENTS];

    // Create socket
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(8080);

    // Bind the socket
    if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
        perror("Socket bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_socket, MAX_CLIENTS) == -1) {
        perror("Socket listen failed");
        exit(EXIT_FAILURE);
    }

    printf("Server listening on port 8080...\n");

    while (1) {
        // Accept a connection
        int client_addr_len = sizeof(client_addr);
        client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &client_addr_len);
        if (client_socket == -1) {
            perror("Error accepting connection");
            continue;
        }
        // Add the client to the clients array
        pthread_mutex_lock(&clients_mutex);
        clients[client_count++] = client_socket;
        pthread_mutex_unlock(&clients_mutex);

        // Create a thread to handle the client
        pthread_create(&client_threads[client_count - 1], NULL, handle_client,
&clients[client_count - 1]);
    }

    // Close the server socket
    close(server_socket);

    return 0;
}


// Client Code

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>

#define MAX_MESSAGE_LEN 1024

void *receive_messages(void *arg) {
    int client_socket = *((int *)arg);
    char message[MAX_MESSAGE_LEN];

    while (1) {
        int receive_size = recv(client_socket, message, MAX_MESSAGE_LEN, 0);
        if (receive_size <= 0) {
            perror("Server disconnected");
            close(client_socket);
            exit(EXIT_FAILURE);
        }

        message[receive_size] = '\0';
        printf("Received: %s", message);
        fflush(stdout);
    }
    pthread_exit(NULL);
}

int main() {
    int client_socket;
    struct sockaddr_in server_addr;
    pthread_t receive_thread;

    // Create socket
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(8080);

    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }

    // Connect to the server
    if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Create a thread to receive messages
    pthread_create(&receive_thread, NULL, receive_messages, &client_socket);

    char message[MAX_MESSAGE_LEN];

    // Send and receive messages
    while (1) {
       // printf("Enter message: ");
        fgets(message, MAX_MESSAGE_LEN, stdin);

        if (send(client_socket, message, strlen(message), 0) < 0) {
            perror("Failed to send message");
            break;
        }
    }

    // Close the client socket
    close(client_socket);

    return 0;
}
```

// LAB-3 Q-2

```
// server_logical_regex_rules_v2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <regex.h>

#define PORT 8080
#define MAX_MESSAGE_LEN 1024

typedef struct {
    char* pattern;
    char* response;
} Rule;

Rule rules[] = {
    {"^(hi|hello|hey).*", "Hello there! How can I assist you today?"},
    {"^(what|tell me about).*(latest laptop|features).*", "Our latest laptop comes with a
high-performance processor, a sleek design, and a long-lasting battery."},
    {"^(what's|tell me about).*(return policy|returns).*", "Our return policy allows for returns within
30 days of purchase. Please visit our website for more details."},
    {"^(what|tell me about).*(latest smartphone|features).*", "Our latest smartphone is the XYZ
model, featuring a high-resolution camera and a long-lasting battery."},
    {"^(how can I contact support|contact information).*", "You can contact our support team at
support@example.com or call us at +1-123-456-7890."},
    {"(business hours|working hours|hours).*\\??$", "Our business hours are Monday to Friday,
9:00 AM to 5:00 PM."},
    {"^(your service is great|you're great|awesome).*", "Thank you for your kind words! We're
here to help. Is there anything specific you'd like assistance with?"},
    {"^(thank you|thanks).*", "You're welcome! If you have any more questions, feel free to ask."},
    {"^(how do I place an order|order process).*", "To place an order, simply visit our website,
select the desired items, and proceed to checkout."},
    {"^(what payment methods do you accept|payment options).*", "We accept credit cards,
PayPal, and other secure payment methods. Your payment information is safe with us."},
    {"^(how do I track my order|order tracking).*", "You can track your order by logging into your
account on our website and navigating to the order tracking section."},
    {"^(what's your warranty policy|product warranty).*", "Our products come with a one-year
warranty. For more details, please refer to our warranty policy on the website."},
    {"^(faq|frequently asked questions).*", "For a list of frequently asked questions, please visit
our FAQ page on the website."},
    {"^(.*bye|goodbye|see you).*", "Goodbye! If you have any more questions in the future, feel
free to reach out."},
    {"^.*$", "I'm sorry, I didn't understand. Could you please provide more details or ask a specific
question?"},
    {NULL, "I'm sorry, I didn't understand. Could you please provide more details or ask a specific
question?"}
};

void handle_intent(char *message, char *response) {
    // Default response (if no match is found)
    strcpy(response, "I'm sorry, I didn't understand. Could you please provide more details or ask
a specific question?");

    // Try to match the message with case-insensitive regular expressions
    for (int i = 0; rules[i].pattern != NULL; i++) {
        regex_t regex;
        if (regcomp(&regex, rules[i].pattern, REG_EXTENDED | REG_NOSUB | REG_ICASE) !=
0) {
            fprintf(stderr, "Failed to compile regex\n");
            continue;
        }

        if (regexec(&regex, message, 0, NULL, 0) == 0) {
            strcpy(response, rules[i].response);
        }

        regfree(&regex);

        if (strcmp(response, "I'm sorry, I didn't understand. Could you please provide more details
or ask a specific question?") != 0) {
            // Stop searching after the first match
            break;
        }
    }
```

```
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    char message[MAX_MESSAGE_LEN];
    char response[MAX_MESSAGE_LEN];

    // Create socket
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Bind the socket
    if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
        perror("Socket bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_socket, 1) == -1) {
        perror("Socket listen failed");
        exit(EXIT_FAILURE);
    }

    printf("Server listening on port %d...\n", PORT);

    // Accept a connection
    int client_addr_len = sizeof(client_addr);
    client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &client_addr_len);
    if (client_socket == -1) {
        perror("Error accepting connection");
        exit(EXIT_FAILURE);
    }

    // Communication loop
    while (1) {
        // Receive client message
        int receive_size = recv(client_socket, message, MAX_MESSAGE_LEN, 0);
        if (receive_size <= 0) {
            perror("Client disconnected");
            break;
        }

        message[receive_size] = '\0';

        // Process intent and generate response
        handle_intent(message, response);

        // Send response to client
        send(client_socket, response, strlen(response), 0);
    }

    // Close sockets
    close(client_socket);
    close(server_socket);

    return 0;
}

// client.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define MAX_MESSAGE_LEN 1024

int main() {
    int client_socket;
    struct sockaddr_in server_addr;
    char message[MAX_MESSAGE_LEN];
    char response[MAX_MESSAGE_LEN];

    // Create socket
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
        perror("Invalid address/Address not supported");
        exit(EXIT_FAILURE);
    }

    // Connect to the server
    if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Communication loop
    while (1) {
        // Get user input
        printf("Enter message: ");
        fgets(message, MAX_MESSAGE_LEN, stdin);

        // Send user input to the server
        send(client_socket, message, strlen(message), 0);

        // Receive and print server response
        int receive_size = recv(client_socket, response, MAX_MESSAGE_LEN, 0);
        if (receive_size <= 0) {
            perror("Server disconnected");
            break;
        }

        response[receive_size] = '\0';
        printf("Server: %s\n", response);
    }

    // Close the client socket
    close(client_socket);

    return 0;
}
```

// LAB-4 Q-1

```
//create sample.txt

// server.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define MAX_BUFFER_SIZE 1024

void receive_file(int client_socket, const char *destination_path) {
    FILE *destination_file = fopen(destination_path, "wb");

    if (destination_file == NULL) {
        perror("Error opening destination file");
        return;
    }

    char buffer[MAX_BUFFER_SIZE];
    size_t bytesRead;

    while ((bytesRead = recv(client_socket, buffer, sizeof(buffer), 0)) > 0) {
        fwrite(buffer, 1, bytesRead, destination_file);
    }

    fclose(destination_file);
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_len = sizeof(client_addr);

    // Create socket
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Bind the socket
    if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
        perror("Socket bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_socket, 1) == -1) {
        perror("Socket listen failed");
        exit(EXIT_FAILURE);
    }

    printf("Server listening on port %d...\n", PORT);

    // Accept a connection
    client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &client_addr_len);
    if (client_socket == -1) {
        perror("Error accepting connection");
        exit(EXIT_FAILURE);
    }

    // Receive file from client and save it
    receive_file(client_socket, "received_file.txt");

    // Close sockets
    close(client_socket);
    close(server_socket);

    return 0;
}

// client.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define MAX_BUFFER_SIZE 1024
```

```
void send_file(int server_socket, const char *source_path) {
    FILE *source_file = fopen(source_path, "rb");

    if (source_file == NULL) {
        perror("Error opening source file");
        return;
    }

    char buffer[MAX_BUFFER_SIZE];
    size_t bytesRead;

    while ((bytesRead = fread(buffer, 1, sizeof(buffer), source_file)) > 0) {
        send(server_socket, buffer, bytesRead, 0);
    }

    fclose(source_file);
}

int main() {
    int client_socket;
    struct sockaddr_in server_addr;

    // Create socket
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");  // Server IP address
    server_addr.sin_port = htons(PORT);

    // Connect to the server
    if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Send file to the server
    send_file(client_socket, "sample.txt");

    // Close the socket
    close(client_socket);

    return 0;
}
```

// LAB-4 Q-2

```
//CAL_server.cpp
#include <bits/stdc++.h>
#include<math.h>
#include <iostream>

#include <cstdlib>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8084

#define PORT1 8080

#define BUFFER_SIZE 1024

using namespace std;


// Function to find precedence of

// operators.

int precedence(char op){

            if(op == '+'||op == '-')

            return 1;

            if(op == '*'||op == '/')

            return 2;
            return 0;

}

int cSocket[5];

void create_scoket(int i){

     struct sockaddr_in serverAddr;


    // Create socket

    if ((cSocket[i] = socket(AF_INET, SOCK_STREAM, 0)) == 0) {

        perror("Socket creation failed");

        exit(EXIT_FAILURE);

    }


    // Set up server address structure

    serverAddr.sin_family = AF_INET;

    serverAddr.sin_port = htons(PORT1+i);


    // Convert IP address from string to binary form

    if (inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr) <= 0) {

        perror("Invalid address/ Address not supported");

        exit(EXIT_FAILURE);

    }


    // Connect to the server

    if (connect(cSocket[i], (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Connection failed");

        exit(EXIT_FAILURE);

    }


    std::cout << "Connected to the server "<<PORT1+i << std::endl;

}

// Function to perform arithmetic operations.

float applyOp(float a, float b, char op){

    int temp=4;

    if(op == '+') temp = 0;

    else if(op == '-') temp = 1;

    else if(op == '*') temp = 2;

    else if(op == '/') temp = 3;



    // Input two numbers from the user


    // Send the numbers to the server

    send(cSocket[temp], &a, sizeof(a), 0);

    send(cSocket[temp], &b, sizeof(b), 0);


    // Receive the sum from the server

    float sum;

    recv(cSocket[temp], &sum, sizeof(sum), 0);

    std::cout << "Sum received from the server: " << sum << std::endl;
    return sum;

}


// Function that returns value of

// expression after evaluation.

float evaluate(string tokens){

            int i;



            // stack to store integer values.

            stack <float> values;



            // stack to store operators.

            stack <char> ops;



            for(i = 0; i < tokens.length(); i++){



                        // Current token is a whitespace,

                        // skip it.

                        if(tokens[i] == ' ')

                                    continue;



                        // Current token is an opening
// brace, push it to 'ops'

else if(tokens[i] == '('){

        ops.push(tokens[i]);

}


// Current token is a number, push

// it to stack for numbers.

else if(isdigit(tokens[i])){

        float val = 0;


        // There may be more than one

        // digits in number.

        while(i < tokens.length() &&

                                isdigit(tokens[i]))

        {

                val = (val*10) + (tokens[i]-'0');

                i++;

        }


        values.push(val);


        // right now the i points to
        // the character next to the digit,

        // since the for loop also increases

        // the i, we would skip one

        // token position; we need to

        // decrease the value of i by 1 to

        // correct the offset.

        i--;

}


// Closing brace encountered, solve

// entire brace.

else if(tokens[i] == ')')

{

        while(!ops.empty() && ops.top() != '(')

        {

                float val2 = values.top();

                values.pop();


                float val1 = values.top();

                values.pop();


                char op = ops.top();
                ops.pop();


                values.push(applyOp(val1, val2, op));
            }


            // pop opening brace.
            if(!ops.empty())
            ops.pop();
        }


        // Current token is an operator.
        else
        {
            // While top of 'ops' has same or greater
            // precedence to current token, which
            // is an operator. Apply operator on top
            // of 'ops' to top two elements in values stack.
            while(!ops.empty() && precedence(ops.top())
                                        >= precedence(tokens[i])){
                float val2 = values.top();
                values.pop();
                float val1 = values.top();
                values.pop();

                char op = ops.top();
                ops.pop();

                values.push(applyOp(val1, val2, op));
            }

            // Push current token to 'ops'.
            ops.push(tokens[i]);
        }
    }

    // Entire expression has been parsed at this
    // point, apply remaining ops to remaining
    // values.
    while(!ops.empty()){
        float val2 = values.top();
        values.pop();

        float val1 = values.top();
```

```
                values.pop();


                char op = ops.top();

                ops.pop();


                values.push(applyOp(val1, val2, op));

            }


            // Top of 'values' contains result, return it.

            return values.top();

    }

    int main() {

        create_scoket(0);

        create_scoket(1);

        create_scoket(2);

        create_scoket(3);

        int serverSocket, clientSocket;

        struct sockaddr_in serverAddr, clientAddr;

        socklen_t addrLen = sizeof(clientAddr);

        // Create socket

        if ((serverSocket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
```

```
    perror("Socket creation failed");

    exit(EXIT_FAILURE);

}

// Set up server address structure

serverAddr.sin_family = AF_INET;

serverAddr.sin_addr.s_addr = INADDR_ANY;

serverAddr.sin_port = htons(PORT);


// Bind the socket to the specified port

if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {

    perror("Bind failed");

    exit(EXIT_FAILURE);

}


// Listen for incoming connections

if (listen(serverSocket, 5) < 0) {

    perror("Listen failed");

    exit(EXIT_FAILURE);

}


std::cout << "cal Server listening on port " << PORT << std::endl;


// Accept a connection from a client

if ((clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddr, &addrLen)) < 0) {
        perror("Accept failed");

        exit(EXIT_FAILURE);

    }


    std::cout << "Connection accepted from " << inet_ntoa(clientAddr.sin_addr) << ":" <<
ntohs(clientAddr.sin_port) << std::endl;


    // Receive two numbers from the client


    while(true){

        char buffer[BUFFER_SIZE] = {0};
        recv(clientSocket, buffer, BUFFER_SIZE, 0);
        string s = buffer;
       // Calculate the sum

       float res = evaluate(s);

       // Send the sum back to the client

       send(clientSocket, &res, sizeof(res), 0);

    }

    // Send the expression to server2

    close(clientSocket);

    // ----- Logic to send expression to server2 -----

    close(cSocket[0]);

    close(cSocket[1]);

    close(cSocket[2]);

    close(cSocket[3]);

    // Create a socket for server2
    return 0;

}


//ADD_server.cpp
#include <iostream>
#include<math.h>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080

int main() {
    int serverSocket, clientSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t addrLen = sizeof(clientAddr);

    // Create socket
    if ((serverSocket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(PORT);

    // Bind the socket to the specified port
    if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(serverSocket, 5) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "ADD_Server listening on port " << PORT << std::endl;

    // Accept a connection from a client
    if ((clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddr, &addrLen)) < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Connection accepted from " << inet_ntoa(clientAddr.sin_addr) << ":" <<
ntohs(clientAddr.sin_port) << std::endl;

    // Receive two numbers from the client
    while(true){
        float num1, num2;
        recv(clientSocket, &num1, sizeof(num1), 0);
        recv(clientSocket, &num2, sizeof(num2), 0);

        // Calculate the sum
        float sum = num1 + num2;

        // Send the sum back to the client
        send(clientSocket, &sum, sizeof(sum), 0);

        std::cout << "Val sent to the cal_server: " << sum << std::endl;
    }

    // Close the sockets
    close(clientSocket);
    close(serverSocket);

    return 0;
}


//SUB_server.cpp
#include <iostream>
#include<math.h>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>
```

```
#define PORT 8081

int main() {
    int serverSocket, clientSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t addrLen = sizeof(clientAddr);

    // Create socket
    if ((serverSocket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(PORT);

    // Bind the socket to the specified port
    if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(serverSocket, 5) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "SUB_Server listening on port " << PORT << std::endl;

    // Accept a connection from a client
    if ((clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddr, &addrLen)) < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Connection accepted from " << inet_ntoa(clientAddr.sin_addr) << ":" <<
ntohs(clientAddr.sin_port) << std::endl;

    // Receive two numbers from the client
    while(true){
        float num1, num2;
        recv(clientSocket, &num1, sizeof(num1), 0);
        recv(clientSocket, &num2, sizeof(num2), 0);

        // Calculate the sum
        float sum = num1 - num2;

        // Send the sum back to the client
        send(clientSocket, &sum, sizeof(sum), 0);

        std::cout << "Val sent to the cal_server: " << sum << std::endl;
    }

    // Close the sockets
    close(clientSocket);
    close(serverSocket);

    return 0;
}

//MUL_server.cpp
#include <iostream>
#include<math.h>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8082

int main() {
    int serverSocket, clientSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t addrLen = sizeof(clientAddr);

    // Create socket
    if ((serverSocket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(PORT);
    // Bind the socket to the specified port
    if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(serverSocket, 5) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "MUL_Server listening on port " << PORT << std::endl;

    // Accept a connection from a client
    if ((clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddr, &addrLen)) < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Connection accepted from " << inet_ntoa(clientAddr.sin_addr) << ":" <<
ntohs(clientAddr.sin_port) << std::endl;

    while(true){
        float num1, num2;
        recv(clientSocket, &num1, sizeof(num1), 0);
        recv(clientSocket, &num2, sizeof(num2), 0);

        // Calculate the sum
        float sum = num1 * num2;

        // Send the sum back to the client
        send(clientSocket, &sum, sizeof(sum), 0);

        std::cout << "Val sent to the cal_server: " << sum << std::endl;
    }

    // Close the sockets
    close(clientSocket);
    close(serverSocket);

    return 0;
}
```

```
//DIV_server.cpp
#include <iostream>
#include<math.h>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8083
using namespace std;
int main() {
    int serverSocket, clientSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t addrLen = sizeof(clientAddr);

    // Create socket
    if ((serverSocket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(PORT);

    // Bind the socket to the specified port
    if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(serverSocket, 5) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Div_Server listening on port " << PORT << std::endl;

    // Accept a connection from a client
    if ((clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddr, &addrLen)) < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Connection accepted from " << inet_ntoa(clientAddr.sin_addr) << ":" <<
ntohs(clientAddr.sin_port) << std::endl;

    // Receive two numbers from the client
    while(true){
        float num1, num2;
        recv(clientSocket, &num1, sizeof(num1), 0);
        recv(clientSocket, &num2, sizeof(num2), 0);

        // Calculate the sum
        float sum ;
        if(num2 == 0){
            cout<<"DIV by 0 "<<endl;
            sum = 0;
        }
        else sum = num1/num2;
        // Send the sum back to the client
        send(clientSocket, &sum, sizeof(sum), 0);

        std::cout << "Val sent to the cal_server: " << sum << std::endl;
    }

    // Close the sockets
    close(clientSocket);
    close(serverSocket);

    return 0;
}


//client.cpp
#include <bits/stdc++.h>
#include<math.h>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8084
using namespace std;
```

```
int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;

    // Create socket
    if ((clientSocket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up server address structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);

    // Convert IP address from string to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }

    // Connect to the server
    if (connect(clientSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Connected to the server" << std::endl;

    while(true){
         // Input two numbers from the user
    string exp;
    cout<<"Enter the expression : ";
    cin>>exp;
    // Send the numbers to the server
    send(clientSocket, exp.c_str(), exp.length(), 0);
    // Receive the sum from the server
    float sum;
    recv(clientSocket, &sum, sizeof(sum), 0);

    std::cout << "Sum received from the server: " << sum << std::endl;

    }
    // Close the socket
    close(clientSocket);
    return 0;
}
```