Bansilal Ramnath Agarwal Charitable Trust's
VISHWAKARMA INSTITUTE OF TECHNOLOGY – PUNE

COMPUTER NETWORKS

Branch	CS - AIML
Division	A
Batch	2
GR-no	12311493
Roll no	54
Name	Atharva Kangralkar

Experiment No. 3b:

TITLE:
Client-Server Programming using UDP Berkeley Socket Primitives.

PROBLEM STATEMENT:

Write the client server programs using UDP Berkeley socket primitives for wired /wireless network for following
a. to say Hello to Each other
b. Calculator (Trigonometry)

THEORY:

This experiment centers on developing a client-server application using the User Datagram Protocol (UDP) with the Berkeley Sockets API.

1. Berkeley Sockets API
The Berkeley Sockets API is a widely used interface for network programming. It provides functions and data structures that enable applications to establish and manage communication over a network. A socket represents one endpoint of this communication, allowing data to be sent or received.

2. User Datagram Protocol (UDP)
UDP is a fundamental protocol within the Internet Protocol (IP) suite. Its main features include:

- Connectionless: Unlike TCP, UDP does not set up a persistent connection before transferring data. Each datagram is transmitted independently and may follow different routes.
- Unreliable: Delivery, order, and integrity of packets are not guaranteed. UDP does not support acknowledgments, retransmissions, or flow control.
- Lightweight: The absence of connection handling and reliability mechanisms makes UDP fast and efficient. It is best suited for applications where low latency is more important than reliability, such as streaming, gaming, and DNS queries.

3. Client-Server Model
The client-server model divides responsibilities between service providers (servers) and requesters (clients).

- Server: Waits for incoming client requests. In this experiment, the server binds to a port, receives datagrams, processes them, and replies back.
- Client: Actively initiates communication by sending requests to the server's IP address and port number.

4. Essential UDP Socket Functions
Key Berkeley socket primitives used in this experiment are:

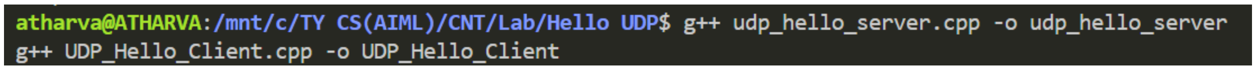
- socket(domain, type, protocol): Creates a new socket.
  - domain: AF_INET indicates IPv4.
  - type: SOCK_DGRAM specifies a UDP (datagram-based) socket.
- bind(sockfd, &addr, addrlen): (Server-side) Assigns the socket to a specific IP address and port so clients can locate the server.
- sendto(sockfd, buffer, len, flags, &dest_addr, addrlen): Sends data from a buffer to a specified destination address.
- recvfrom(sockfd, buffer, len, flags, &src_addr, &addrlen): Receives data into a buffer while also recording the sender's address, allowing the server to reply.
- close(sockfd): Terminates the socket and frees system resource

PROCEDURE:

The following steps outline how to compile and run the client-server programs in a Linux/Unix environment. You will need two separate terminal windows: one for the server and one for the client.

Part a: Hello to Each other

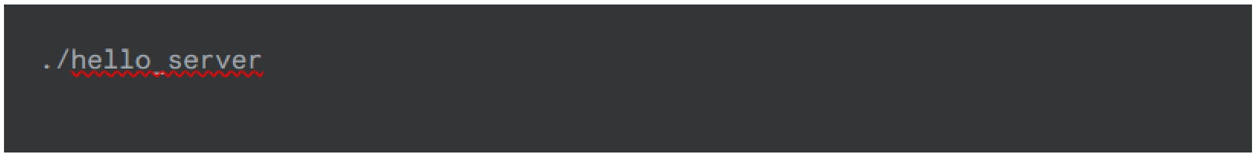
Compilation:
Open a terminal and navigate to the directory where you saved the files udp_hello_server.cpp and udp_hello_client.cpp. Compile the programs using the g++ compiler:



```
atharva@ATHARVA:/mnt/c/TY CS(AIML)/CNT/Lab/Hello UDP$ g++ udp_hello_server.cpp -o udp_hello_server
g++ UDP_Hello_Client.cpp -o UDP_Hello_Client
```

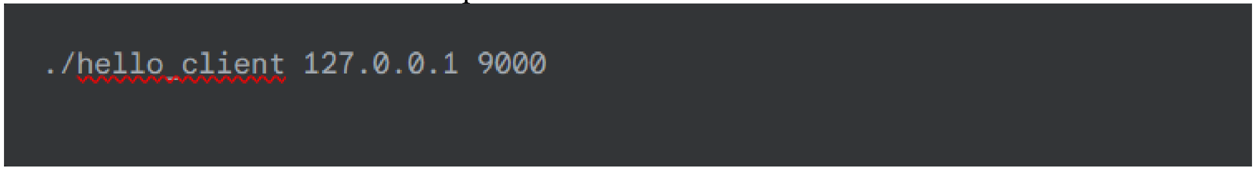
Execution:

- **Terminal 1 (Server):** Run the compiled server executable. It will wait for a message from a client.



```
./hello_server
```

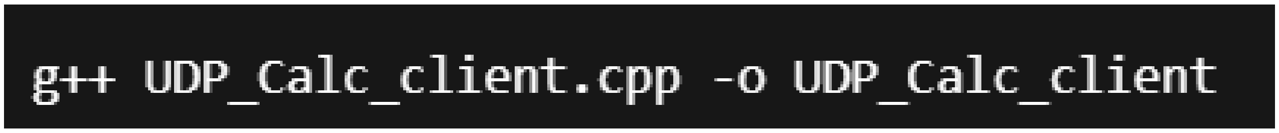
- **Terminal 2 (Client):** Run the compiled client executable, providing the server's IP address and port number as command-line arguments. If you are running both on the same machine, use 127.0.0.1 as the IP. The port is 9000 as defined in the server code.



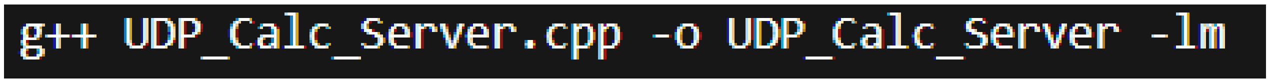
```
./hello_client 127.0.0.1 9000
```

Part b: Calculator (Trigonometry)

Compilation:
Open a terminal and navigate to the directory where you saved udp_calc_server.cpp and udp_calc_client.cpp. Compile the programs, making sure to link the math library (-lm) for the server code.



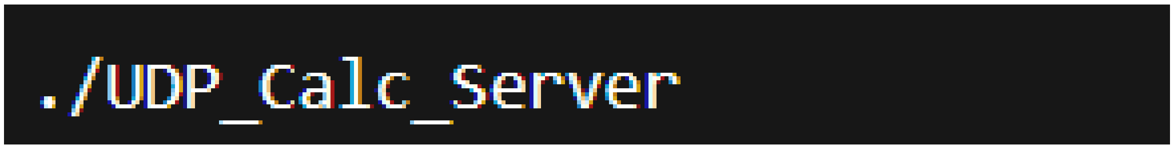
```
g++ UDP_Calc_client.cpp -o UDP_Calc_client
```



```
g++ UDP_Calc_Server.cpp -o UDP_Calc_Server -lm
```

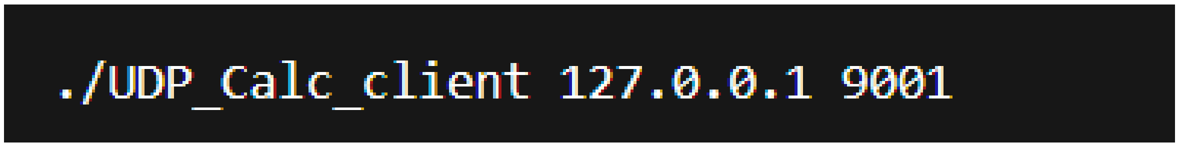
Execution:

- **Terminal 1 (Server):** Run the compiled calculator server. It will start and wait for expressions from the client.



```
./UDP_Calc_Server
```

- **Terminal 2 (Client):** Run the compiled calculator client, providing the server's IP (127.0.0.1 for local) and port (9001).

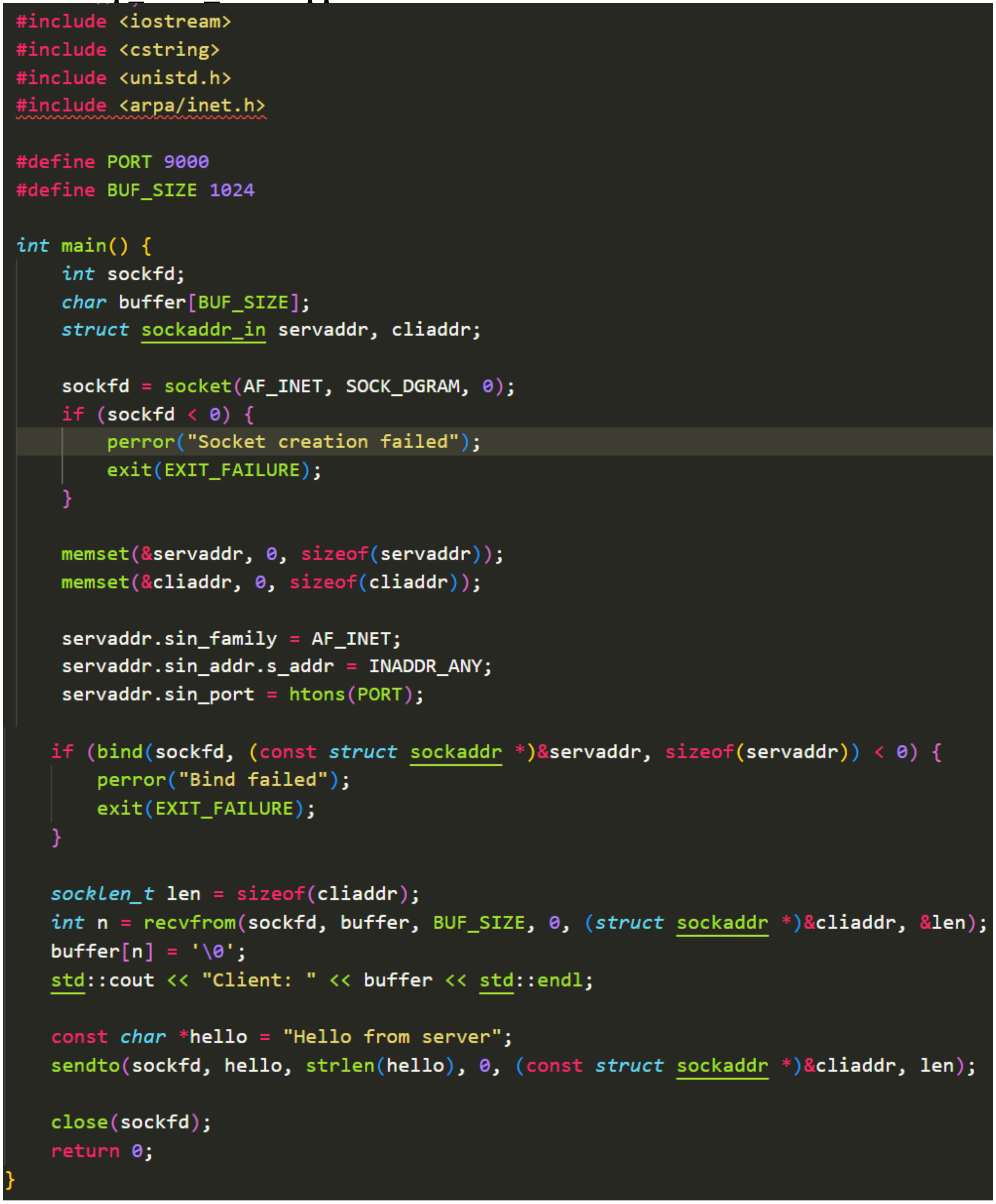


```
./UDP_Calc_client 127.0.0.1 9001
```

- You can now enter trigonometric expressions in the client terminal (e.g., sin 30 deg, cos 1.57 rad). Type quit to exit the client and server.

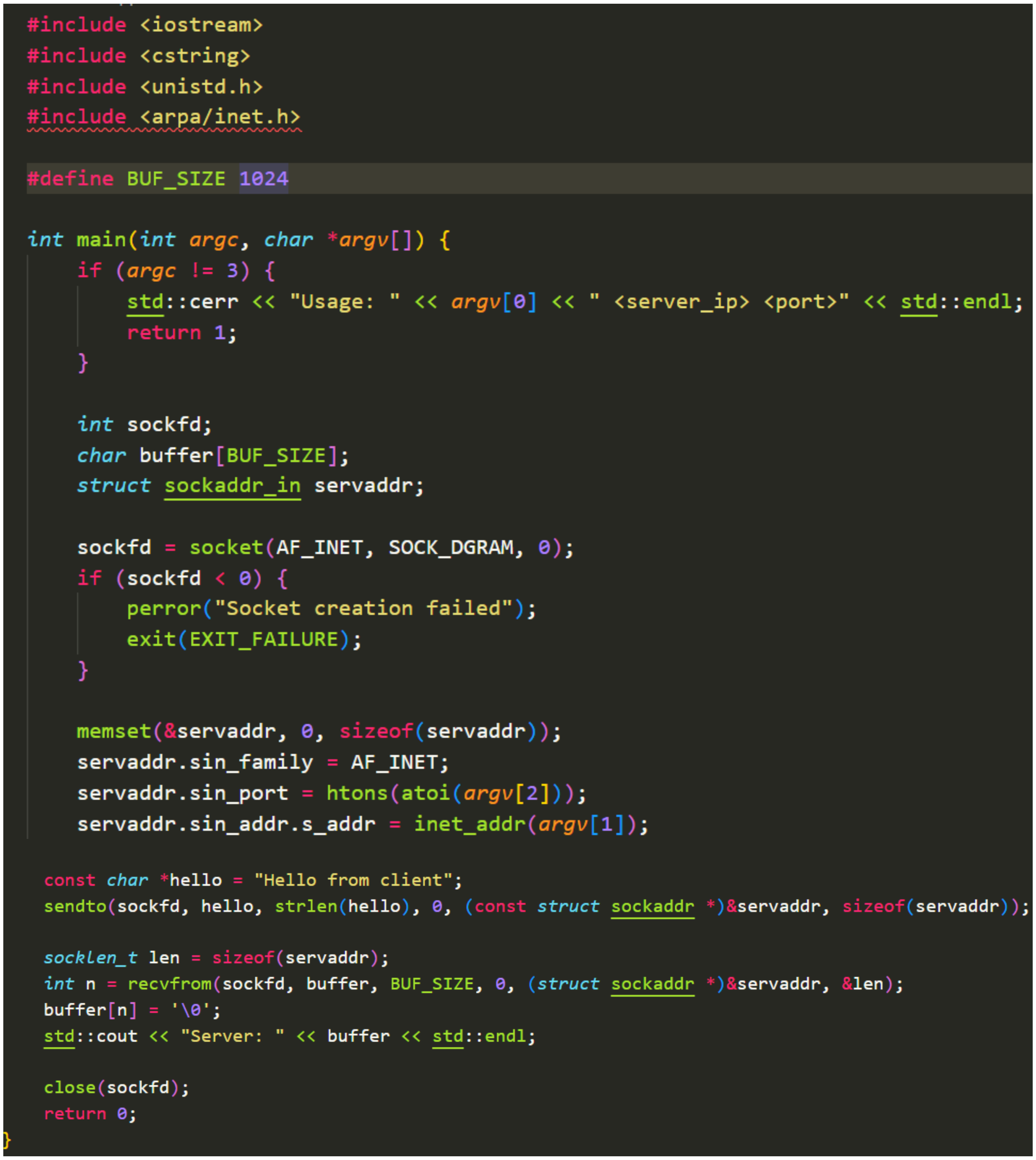
A. Hello message exchange

1. udp_hello_sever.cpp



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

#define PORT 9000
#define BUF_SIZE 1024

int main() {
    int sockfd;
    char buffer[BUF_SIZE];
    struct sockaddr_in servaddr, cliaddr;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);

   if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
       perror("Bind failed");
       exit(EXIT_FAILURE);
   }

   socklen_t len = sizeof(cliaddr);
   int n = recvfrom(sockfd, buffer, BUF_SIZE, 0, (struct sockaddr *)&cliaddr, &len);
   buffer[n] = '\0';
   std::cout << "Client: " << buffer << std::endl;

   const char *hello = "Hello from server";
   sendto(sockfd, hello, strlen(hello), 0, (const struct sockaddr *)&cliaddr, len);

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

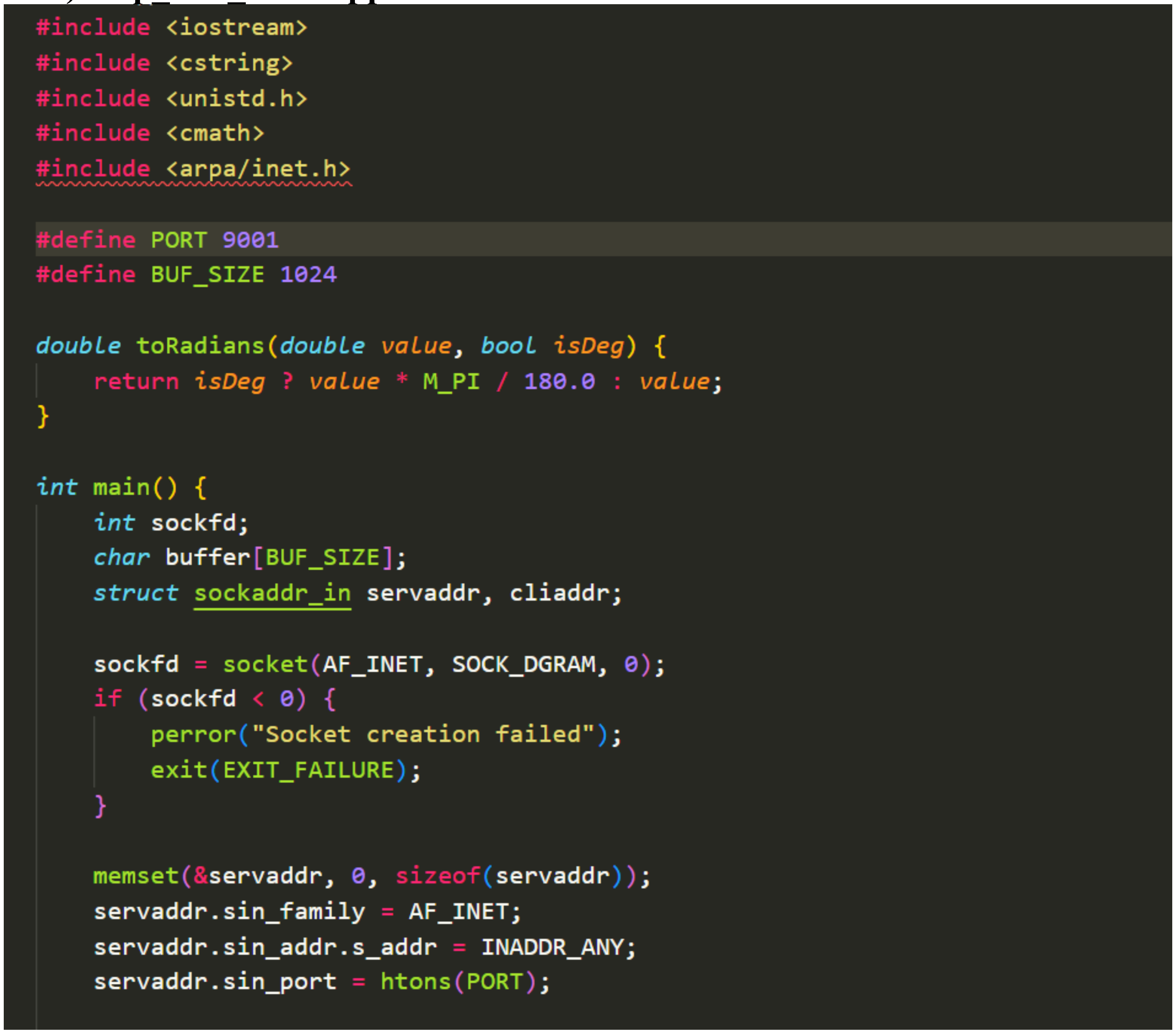
2. udp_hello_client.cpp



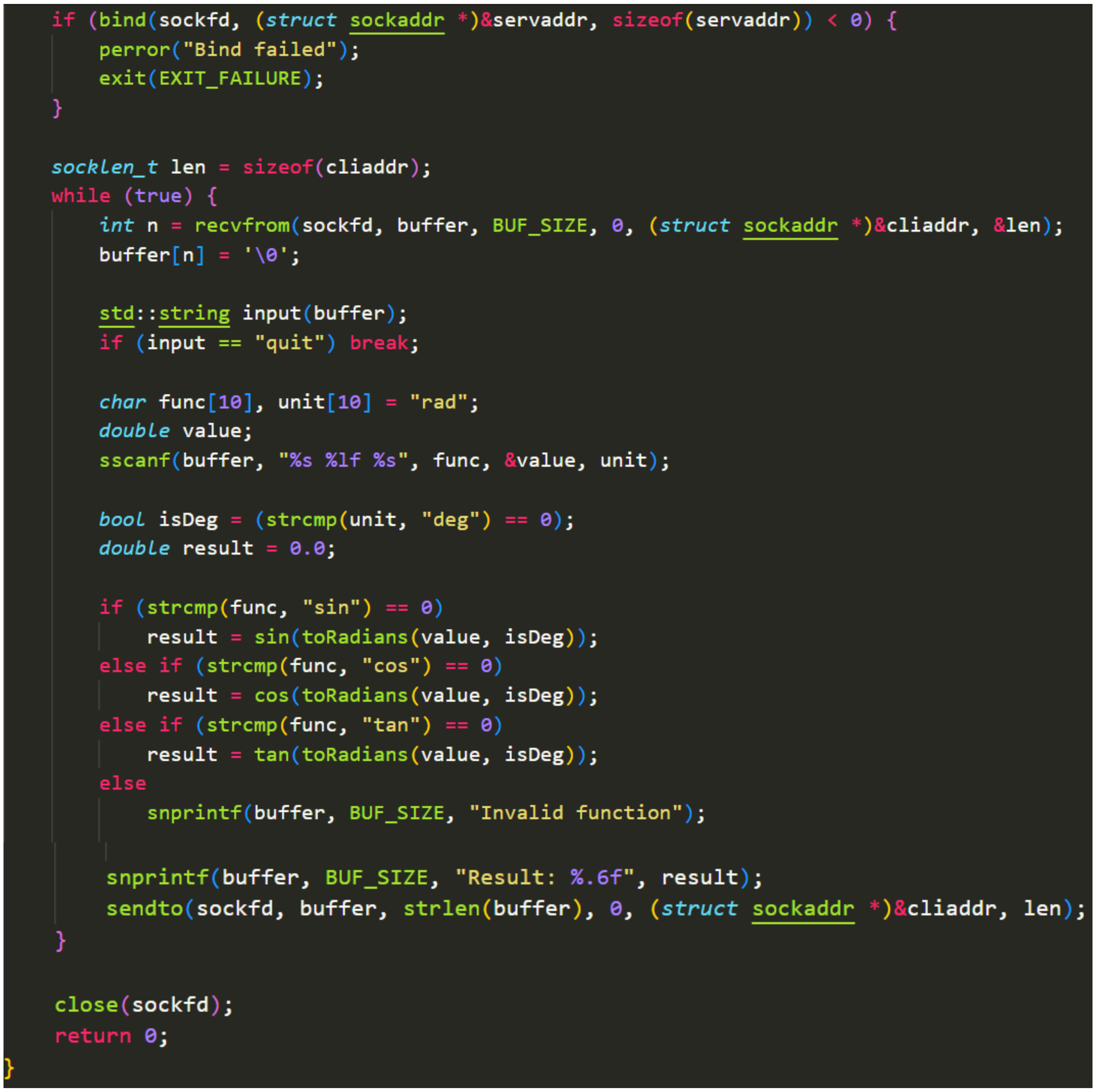
```cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define BUF_SIZE 1024

int main(int argc, char *argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " <server_ip> <port>" << std::endl;
        return 1;
    }

    int sockfd;
    char buffer[BUF_SIZE];
    struct sockaddr_in servaddr;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(atoi(argv[2]));
    servaddr.sin_addr.s_addr = inet_addr(argv[1]);

    const char *hello = "Hello from client";
    sendto(sockfd, hello, strlen(hello), 0, (const struct sockaddr *)&servaddr, sizeof(servaddr));

    socklen_t len = sizeof(servaddr);
    int n = recvfrom(sockfd, buffer, BUF_SIZE, 0, (struct sockaddr *)&servaddr, &len);
    buffer[n] = '\0';
    std::cout << "Server: " << buffer << std::endl;

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

B. Calculator (Trigonometry)

1) udp_calc_sever.cpp

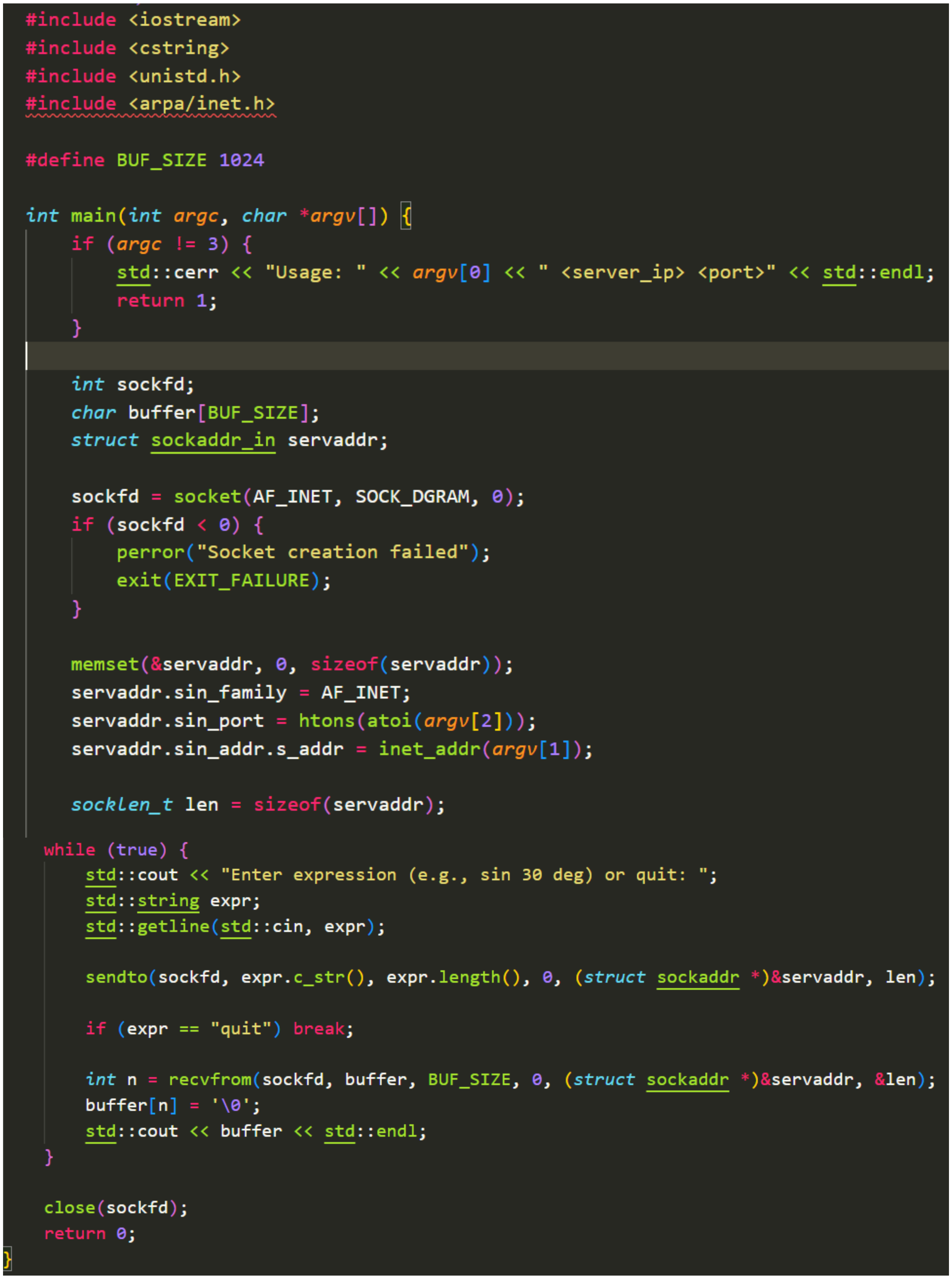


```cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <cmath>
#include <arpa/inet.h>

#define PORT 9001
#define BUF_SIZE 1024

double toRadians(double value, bool isDeg) {
    return isDeg ? value * M_PI / 180.0 : value;
}

int main() {
    int sockfd;
    char buffer[BUF_SIZE];
    struct sockaddr_in servaddr, cliaddr;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);
```



```cpp
    if (bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    socklen_t len = sizeof(cliaddr);
    while (true) {
        int n = recvfrom(sockfd, buffer, BUF_SIZE, 0, (struct sockaddr *)&cliaddr, &len);
        buffer[n] = '\0';

        std::string input(buffer);
        if (input == "quit") break;

        char func[10], unit[10] = "rad";
        double value;
        sscanf(buffer, "%s %lf %s", func, &value, unit);

        bool isDeg = (strcmp(unit, "deg") == 0);
        double result = 0.0;

        if (strcmp(func, "sin") == 0)
            result = sin(toRadians(value, isDeg));
        else if (strcmp(func, "cos") == 0)
            result = cos(toRadians(value, isDeg));
        else if (strcmp(func, "tan") == 0)
            result = tan(toRadians(value, isDeg));
        else
            snprintf(buffer, BUF_SIZE, "Invalid function");

        snprintf(buffer, BUF_SIZE, "Result: %.6f", result);
        sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)&cliaddr, len);
    }

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

2) udp_calc_client.cpp

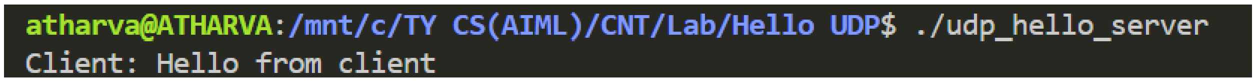


```
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define BUF_SIZE 1024

int main(int argc, char *argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " <server_ip> <port>" << std::endl;
        return 1;
    }

    int sockfd;
    char buffer[BUF_SIZE];
    struct sockaddr_in servaddr;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(atoi(argv[2]));
    servaddr.sin_addr.s_addr = inet_addr(argv[1]);

    socklen_t len = sizeof(servaddr);

  while (true) {
      std::cout << "Enter expression (e.g., sin 30 deg) or quit: ";
      std::string expr;
      std::getline(std::cin, expr);

      sendto(sockfd, expr.c_str(), expr.length(), 0, (struct sockaddr *)&servaddr, len);

      if (expr == "quit") break;

      int n = recvfrom(sockfd, buffer, BUF_SIZE, 0, (struct sockaddr *)&servaddr, &len);
      buffer[n] = '\0';
      std::cout << buffer << std::endl;
  }

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

SCREENSHOTS/OUTPUT:

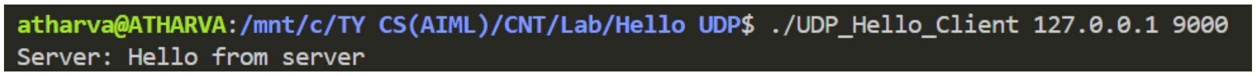
A. **Hello Program Output**

i. **Terminal 1: Server Output**



```
atharva@ATHARVA:/mnt/c/TY CS(AIML)/CNT/Lab/Hello UDP$ ./udp_hello_server
Client: Hello from client
```

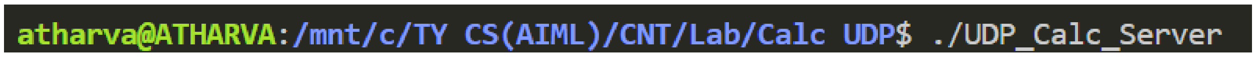
ii. **Terminal 2: Client Output**



```
atharva@ATHARVA:/mnt/c/TY CS(AIML)/CNT/Lab/Hello UDP$ ./UDP_Hello_Client 127.0.0.1 9000
Server: Hello from server
```

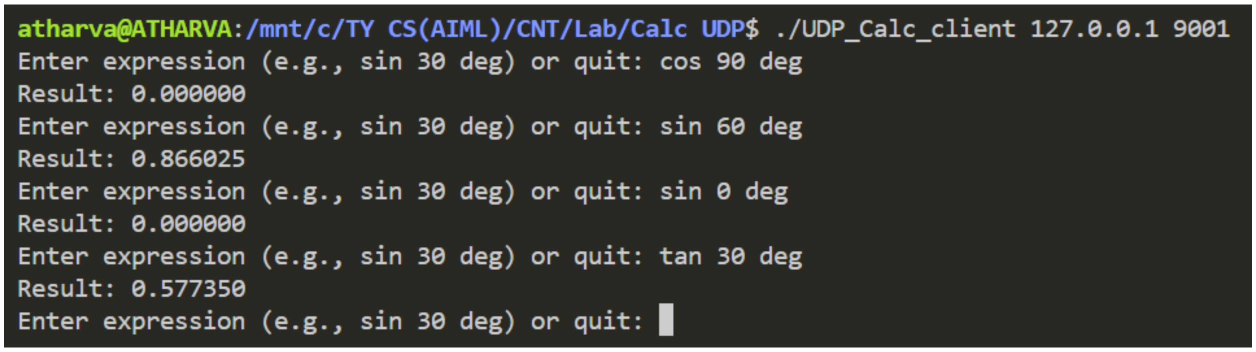
B. **Calculator Program Output**

iii. **Terminal 1: Server Output**



```
atharva@ATHARVA:/mnt/c/TY CS(AIML)/CNT/Lab/Calc UDP$ ./UDP_Calc_Server
```

iv. **Terminal 2: Client Output**



```
atharva@ATHARVA:/mnt/c/TY CS(AIML)/CNT/Lab/Calc UDP$ ./UDP_Calc_client 127.0.0.1 9001
Enter expression (e.g., sin 30 deg) or quit: cos 90 deg
Result: 0.000000
Enter expression (e.g., sin 30 deg) or quit: sin 60 deg
Result: 0.866025
Enter expression (e.g., sin 30 deg) or quit: sin 0 deg
Result: 0.000000
Enter expression (e.g., sin 30 deg) or quit: tan 30 deg
Result: 0.577350
Enter expression (e.g., sin 30 deg) or quit: 
```

CONCLUSION:

This experiment successfully illustrated how to implement a client-server model using the User Datagram Protocol (UDP) with Berkeley socket primitives. The main insights gained from the assignment are:

1. UDP as a Connectionless Protocol: Unlike TCP, UDP does not maintain a continuous connection. Each message (datagram) is sent independently, as demonstrated by the use of sendto() and recvfrom() for transmitting and receiving individual packets.
2. Basics of Socket Programming: The exercise reinforced the essential steps in socket programming:
    - socket(): Creating a communication endpoint.
    - bind(): Linking the server socket to a specific port and address so clients know

    where to connect.
    - sendto() & recvfrom(): Core functions for exchanging data in UDP, requiring the destination/source address each time.
    - close(): Releasing the socket resource once communication ends.
3. Roles of Client and Server: The difference between the two roles was made clear. The server passively binds to a port and listens, whereas the client actively sends a message to the server's predefined address and port.
4. Practical Implementation: Two applications were built—a basic "Hello" message exchange and a trigonometric calculator. This highlighted how the same UDP structure can support different services simply by modifying the transmitted data and the server-side logic.

In summary, the assignment offered valuable hands-on practice in developing network applications and deepened understanding of the stateless, connectionless nature of UDP communication.