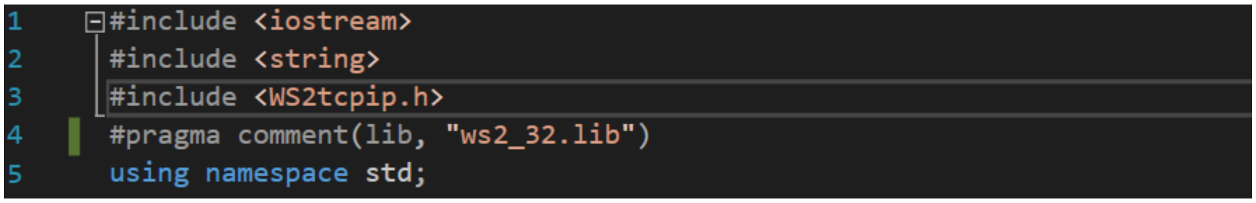
Project 1: Client-Server Chat

Created by Luis Umana

TCP Client and Server:

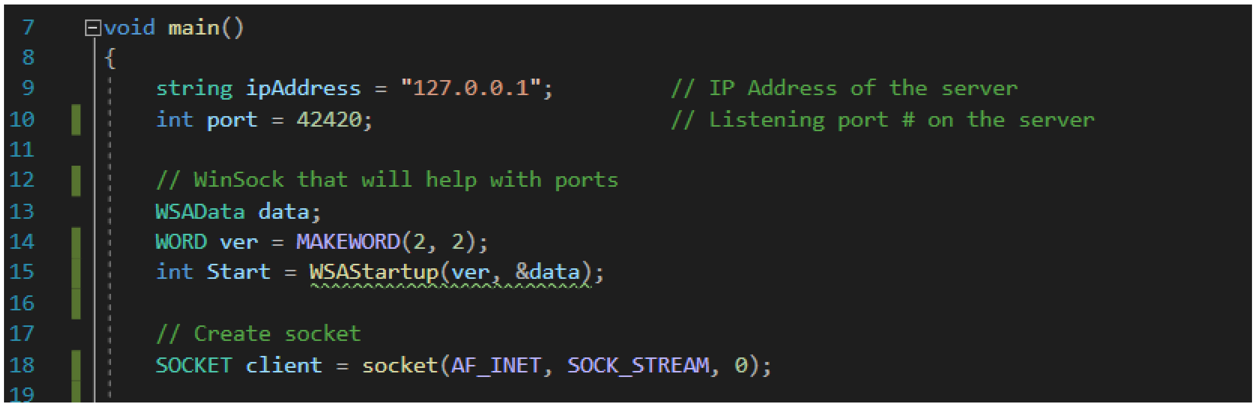
TCP Client:

The client was created in C++ using Visual Studio as the complier, and WinSock for the sockets. The libraries include some basic C++ libraries as well as libraries to help WinSock work more easily, WS2tcpip.h and ws2_32.lb.



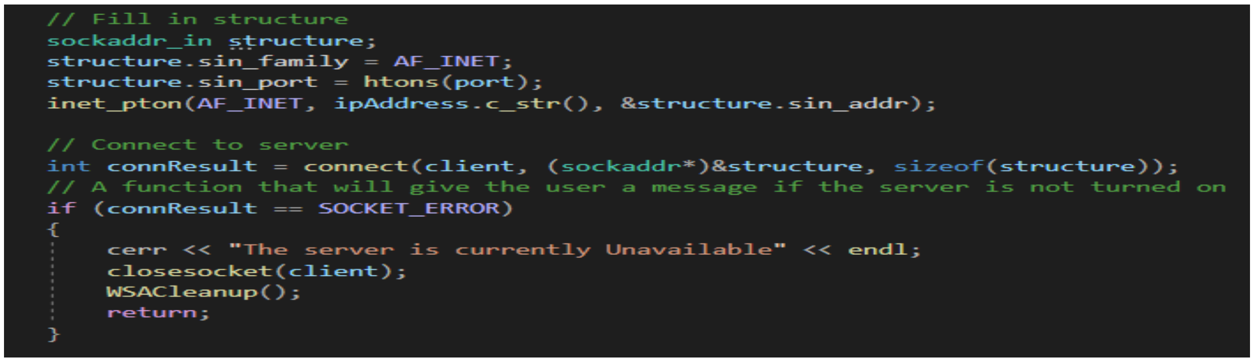
```
#include <iostream>
#include <string>
#include <WS2tcpip.h>
#pragma comment(lib, "ws2_32.lib")
using namespace std;
```

After setting up the ip-address and a functioning port, I inserted a Winsock function that allows me to have easier access to the ports and I created a socket for the client.

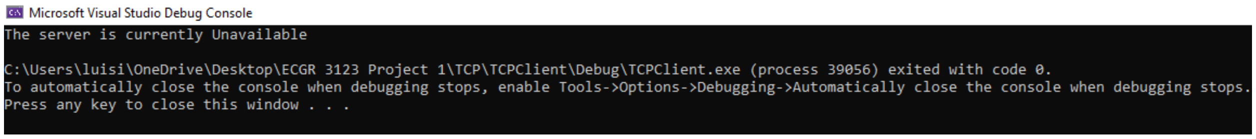


```
void main()
{
    string ipAddress = "127.0.0.1";         // IP Address of the server
    int port = 42420;                       // Listening port # on the server

    // WinSock that will help with ports
    WSAData data;
    WORD ver = MAKEWORD(2, 2);
    int Start = WSAStartup(ver, &data);

    // Create socket
    SOCKET client = socket(AF_INET, SOCK_STREAM, 0);

```

Once the socket has been created, I also created the address structure using the port and ip-address. Afterwards, I created a basic connect code that uses the address structure that allows the client to attempt the connect to the server. In case the client cannot connect to the server; whether it be a wrong port, or the server is off, a message will be sent to the user to tell them the server is unavailable.



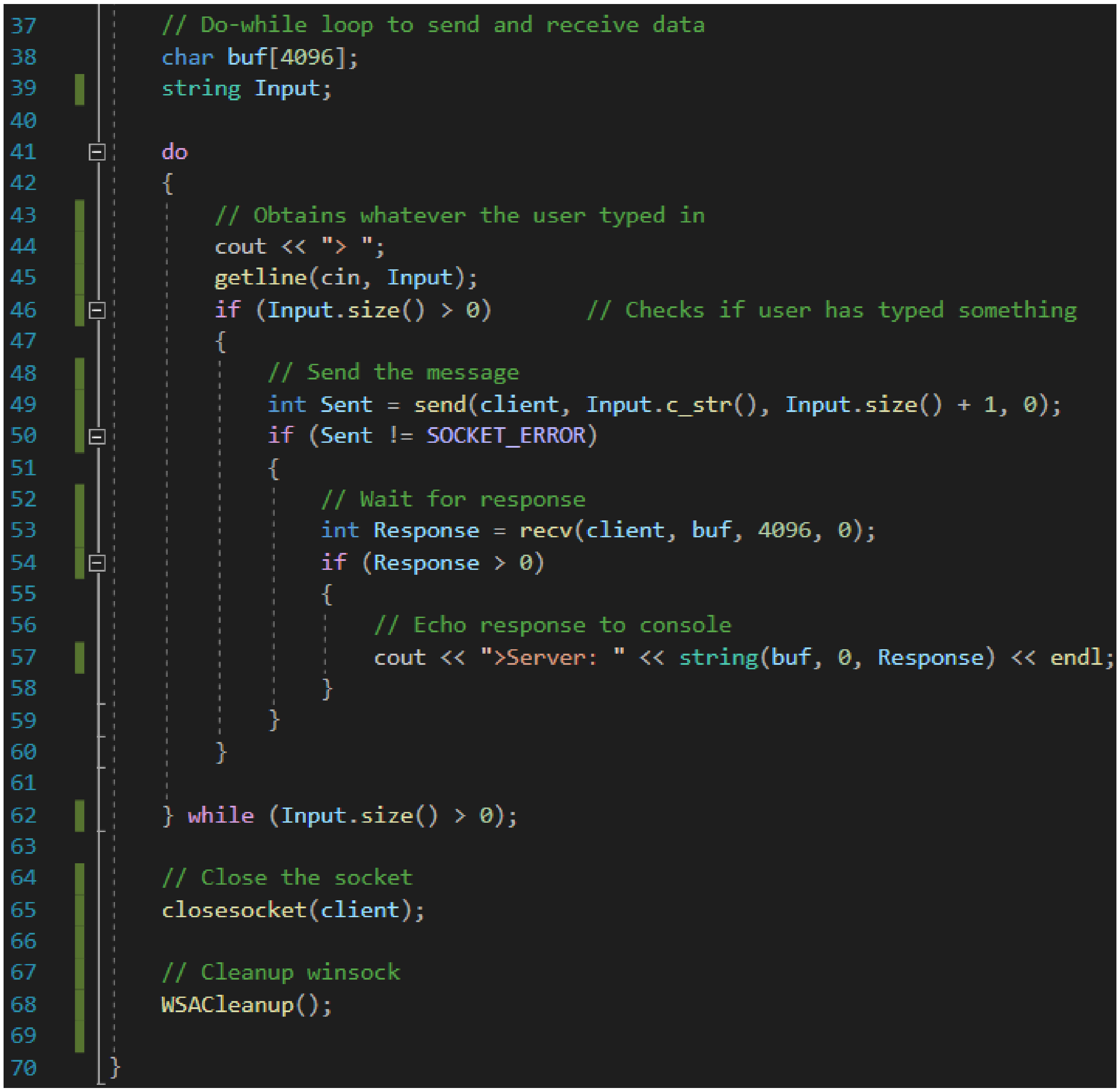
```
// Fill in structure
sockaddr_in structure;
structure.sin_family = AF_INET;
structure.sin_port = htons(port);
inet_pton(AF_INET, ipAddress.c_str(), &structure.sin_addr);

// Connect to server
int connResult = connect(client, (sockaddr*)&structure, sizeof(structure));
// A function that will give the user a message if the server is not turned on
if (connResult == SOCKET_ERROR)
{
    cerr << "The server is currently Unavailable" << endl;
    closesocket(client);
    WSACleanup();
    return;
}
```



Microsoft Visual Studio Debug Console

```
The server is currently Unavailable

C:\Users\luisi\OneDrive\Desktop\ECGR 3123 Project 1\TCP\TCPClient\Debug\TCPClient.exe (process 39056) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

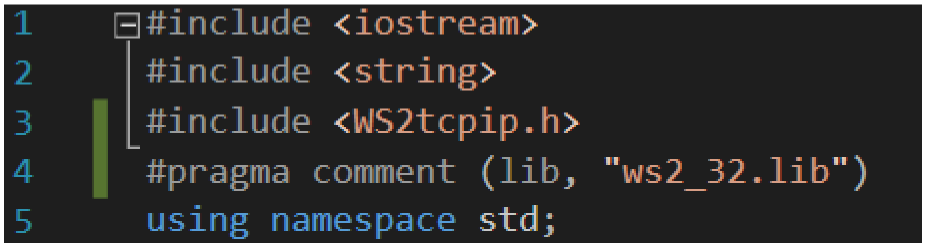
The do while loop does the main work for the client. It will ask the user for an input once they have connected to the server. They will then send the message to the server and wait for the server to respond back. Once the server has sent a message back, it will indicate to the client that it came from server in line 57. This conversation will repeat until the client leaves or the server stops functioning.



```
37      // Do-while loop to send and receive data
38      char buf[4096];
39      string Input;
40
41      do
42      {
43          // Obtains whatever the user typed in
44          cout << "> ";
45          getline(cin, Input);
46          if (Input.size() > 0)       // Checks if user has typed something
47          {
48              // Send the message
49              int Sent = send(client, Input.c_str(), Input.size() + 1, 0);
50              if (Sent != SOCKET_ERROR)
51              {
52                  // Wait for response
53                  int Response = recv(client, buf, 4096, 0);
54                  if (Response > 0)
55                  {
56                      // Echo response to console
57                      cout << ">Server: " << string(buf, 0, Response) << endl;
58                  }
59              }
60          }
61
62      } while (Input.size() > 0);
63
64      // Close the socket
65      closesocket(client);
66
67      // Cleanup winsock
68      WSACleanup();
69
70  }
```

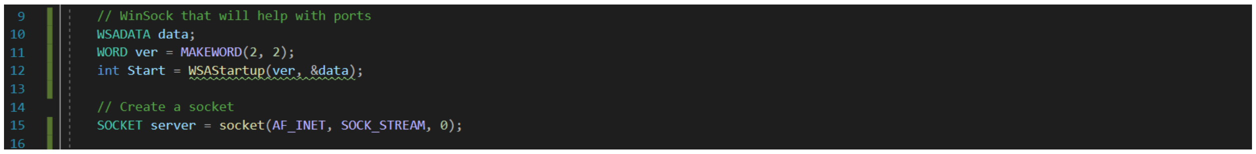
TCP Server:

As mentioned in the client portion of the paper, the server is also made in visual studio, uses Winsock, and has the same libraries as the client.



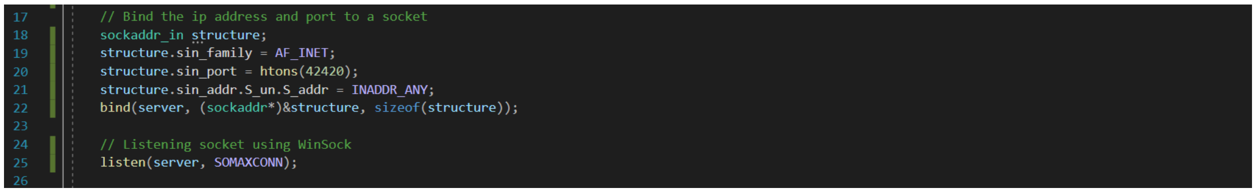
```
#include <iostream>
#include <string>
#include <WS2tcpip.h>
#pragma comment (lib, "ws2_32.lib")
using namespace std;
```

Once again, I created a function to let Winsock work and a socket for server this time



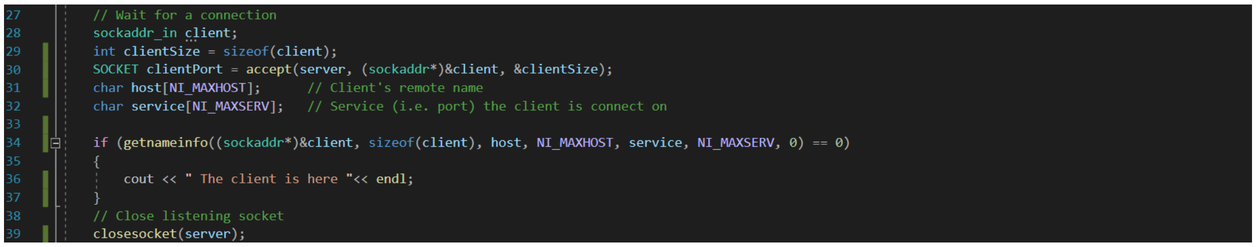
```
// WinSock that will help with ports
WSADATA data;
WORD ver = MAKEWORD(2, 2);
int Start = WSAStartup(ver, &data);

// Create a socket
SOCKET server = socket(AF_INET, SOCK_STREAM, 0);
```

After creating the address for the server, I bind the address and port to the client socket, and the listen command is used to wait for a connection from the client.



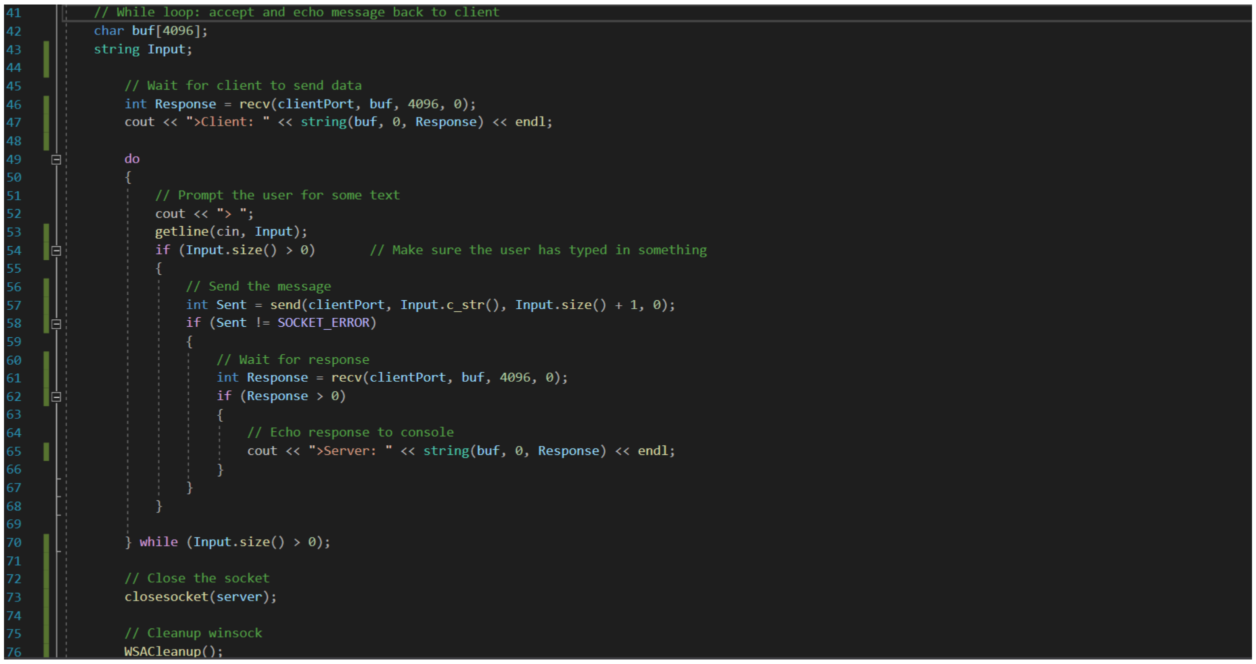
```
// Bind the ip address and port to a socket
sockaddr_in structure;
structure.sin_family = AF_INET;
structure.sin_port = htons(42420);
structure.sin_addr.S_un.S_addr = INADDR_ANY;
bind(server, (sockaddr*)&structure, sizeof(structure));

// Listening socket using WinSock
listen(server, SOMAXCONN);
```

As the server is waiting for a connection, it will try to obtain the information needed from the client. Once the client is connected to the server, a message will be displayed to the server that the server is here. (As shown in the final results.)



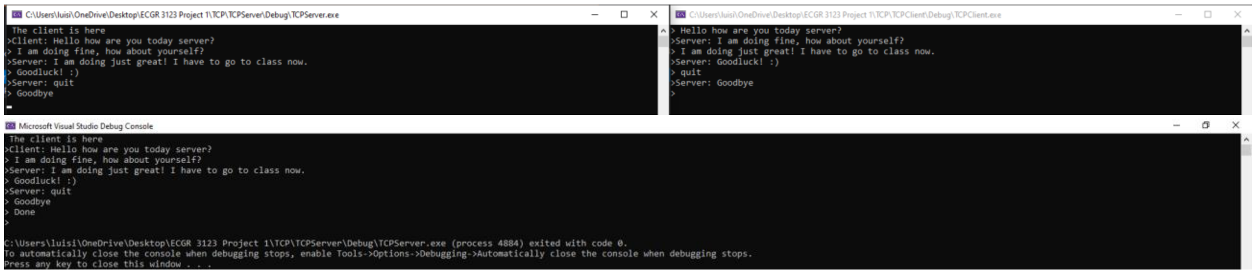
```
// Wait for a connection
sockaddr_in client;
int clientSize = sizeof(client);
SOCKET clientPort = accept(server, (sockaddr*)&client, &clientSize);
char host[NI_MAXHOST];      // Client's remote name
char service[NI_MAXSERV];   // Service (i.e. port) the client is connect on

if (getnameinfo((sockaddr*)&client, sizeof(client), host, NI_MAXHOST, service, NI_MAXSERV, 0) == 0)
{
    cout << " The client is here "<< endl;
}
// Close listening socket
closesocket(server);
```

The server is a bit different in the chat, as the server has to wait for the client to make the initial contact. Once the client has sent a message, line 47 will display the message to the server and will tell it that it came from the client. The do while loop message system is the exact same as the client



```cpp
41      // While loop: accept and echo message back to client
42      char buf[4096];
43      string Input;
44
45          // Wait for client to send data
46          int Response = recv(clientPort, buf, 4096, 0);
47          cout << ">Client: " << string(buf, 0, Response) << endl;
48
49          do
50          {
51              // Prompt the user for some text
52              cout << "> ";
53              getline(cin, Input);
54              if (Input.size() > 0)       // Make sure the user has typed in something
55              {
56                  // Send the message
57                  int Sent = send(clientPort, Input.c_str(), Input.size() + 1, 0);
58                  if (Sent != SOCKET_ERROR)
59                  {
60                      // Wait for response
61                      int Response = recv(clientPort, buf, 4096, 0);
62                      if (Response > 0)
63                      {
64                          // Echo response to console
65                          cout << ">Server: " << string(buf, 0, Response) << endl;
66                      }
67                  }
68              }
69
70          } while (Input.size() > 0);
71
72          // Close the socket
73          closesocket(server);
74
75          // Cleanup winsock
76          WSACleanup();
```

TCP Client and Server Final Result:



```
C:\Users\luisi\OneDrive\Desktop\ECGR 3123 Project 1\TCP\TCPServer\Debug\TCPServer.exe
 The client is here
>Client: Hello how are you today server?
> I am doing fine, how about yourself?
>Server: I am doing just great! I have to go to class now.
> Goodluck! :)
>Server: quit
> Goodbye
```

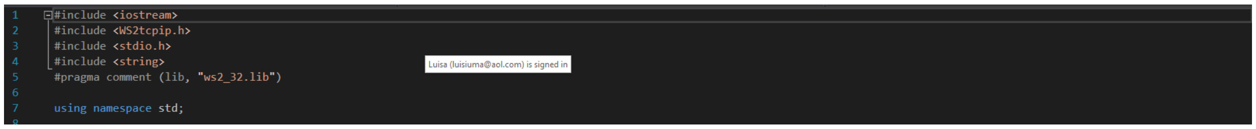
```
C:\Users\luisi\OneDrive\Desktop\ECGR 3123 Project 1\TCP\TCPClient\Debug\TCPClient.exe
> Hello how are you today server?
>Server: I am doing fine, how about yourself?
> I am doing just great! I have to go to class now.
>Server: Goodluck! :)
> quit
>Server: Goodbye
>
```

```
Microsoft Visual Studio Debug Console
 The client is here
>Client: Hello how are you today server?
> I am doing fine, how about yourself?
>Server: I am doing just great! I have to go to class now.
> Goodluck! :)
>Server: quit
> Goodbye
> Done
>

C:\Users\luisi\OneDrive\Desktop\ECGR 3123 Project 1\TCP\TCPServer\Debug\TCPServer.exe (process 4884) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

UDP Client and Server:

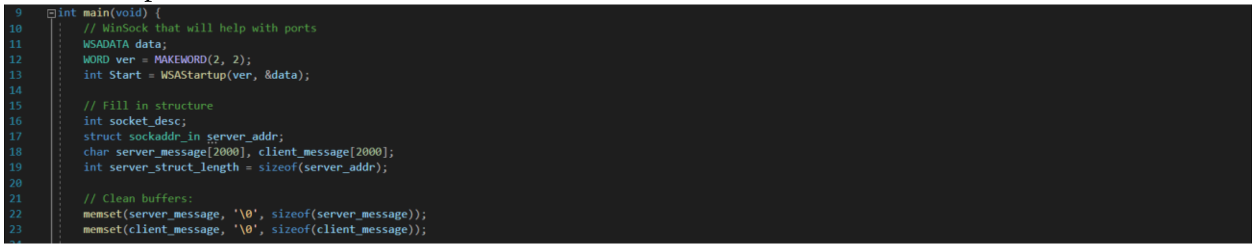
UDP Client:

I had a very tough time trying to do UDP in C++, so I used C instead, however it has the same libraries and complier.



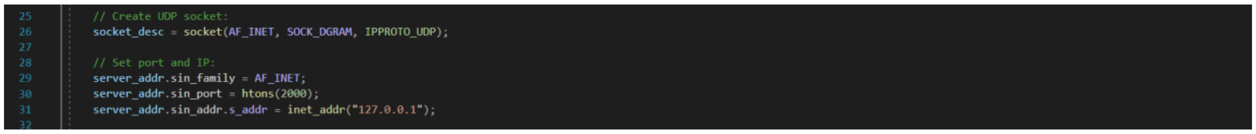
```
#include <iostream>
#include <WS2tcpip.h>
#include <stdio.h>
#include <string>
#pragma comment (lib, "ws2_32.lib")

using namespace std;
```

Just like TCP, it has a function for Winsock to help with the ports and the address for the client was created. The clean buffers prevent the system from sending symbols by clearing the buffer before it repeats the next action.



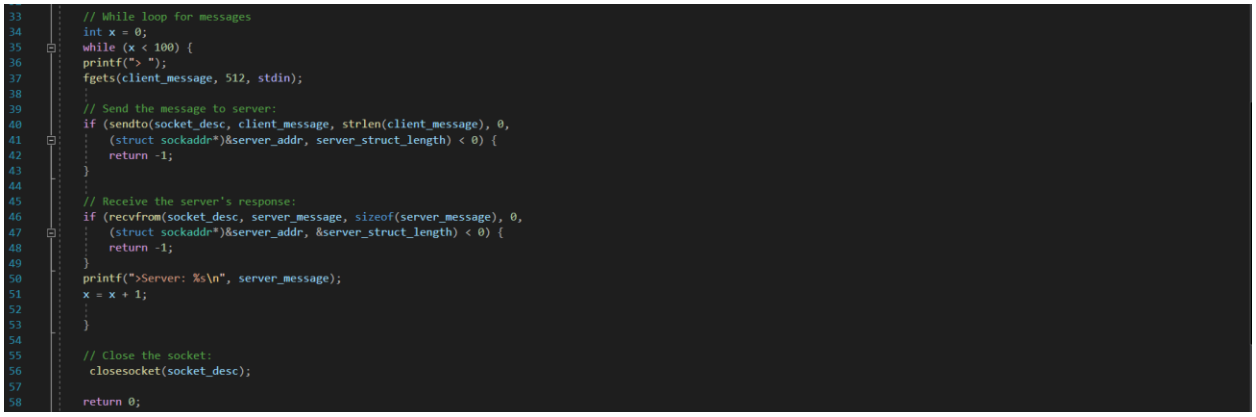
```
int main(void) {
    // WinSock that will help with ports
    WSADATA data;
    WORD ver = MAKEWORD(2, 2);
    int Start = WSAStartup(ver, &data);

    // Fill in structure
    int socket_desc;
    struct sockaddr_in server_addr;
    char server_message[2000], client_message[2000];
    int server_struct_length = sizeof(server_addr);

    // Clean buffers:
    memset(server_message, '\0', sizeof(server_message));
    memset(client_message, '\0', sizeof(client_message));
```

The socket was then created, and a different port was used in case it interferes with the TCP communication.



```
    // Create UDP socket:
    socket_desc = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    // Set port and IP:
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(2000);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

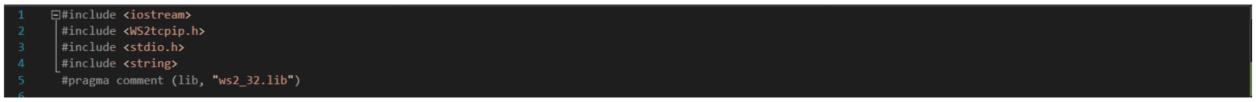
Finally, a while loop was created for the send and receive messages. The client would make the initial contact and send whatever message they , wait for the response of the server, and



```
    // While loop for messages
    int x = 0;
    while (x < 100) {
    printf("> ");
    fgets(client_message, 512, stdin);

    // Send the message to server:
    if (sendto(socket_desc, client_message, strlen(client_message), 0,
        (struct sockaddr*)&server_addr, server_struct_length) < 0) {
        return -1;
    }

    // Receive the server's response:
    if (recvfrom(socket_desc, server_message, sizeof(server_message), 0,
        (struct sockaddr*)&server_addr, &server_struct_length) < 0) {
        return -1;
    }
    printf(">Server: %s\n", server_message);
    x = x + 1;

    }

    // Close the socket:
     closesocket(socket_desc);

    return 0;
```

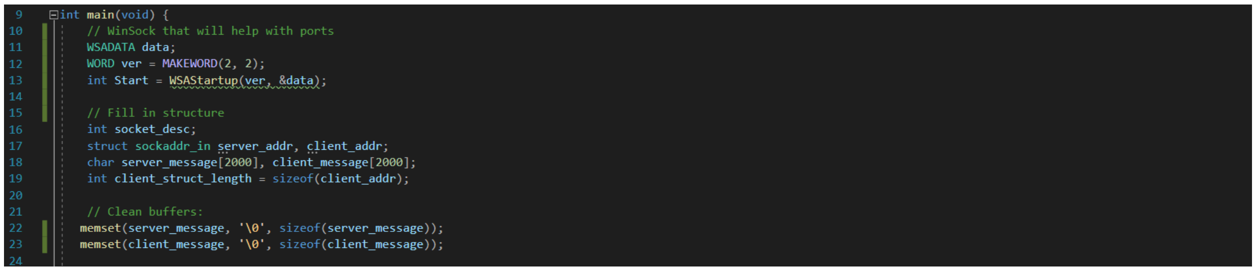
UDP Server:

Here are the libraries used for the server, exactly the same to the Client and TCP files.



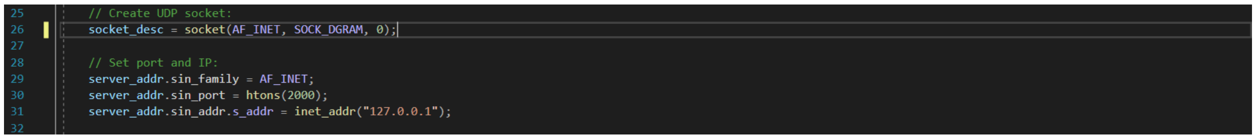
```
#include <iostream>
#include <WS2tcpip.h>
#include <stdio.h>
#include <string>
#pragma comment (lib, "ws2_32.lib")
```

Just like TCP and the Client file, it has a function for Winsock, address for the server, and clean buffers.

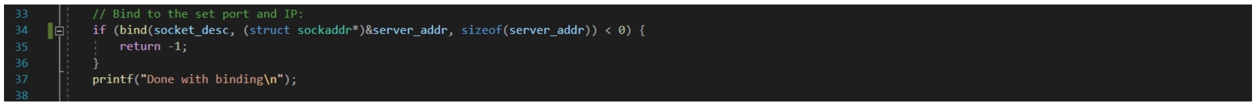


```
int main(void) {
    // WinSock that will help with ports
    WSADATA data;
    WORD ver = MAKEWORD(2, 2);
    int Start = WSAStartup(ver, &data);

    // Fill in structure
    int socket_desc;
    struct sockaddr_in server_addr, client_addr;
    char server_message[2000], client_message[2000];
    int client_struct_length = sizeof(client_addr);

    // Clean buffers:
   memset(server_message, '\0', sizeof(server_message));
   memset(client_message, '\0', sizeof(client_message));
```

The socket was then created and has the same port as the client. The server would then bind structure and socket information. The server would send a message to itself once it is connected to the server.

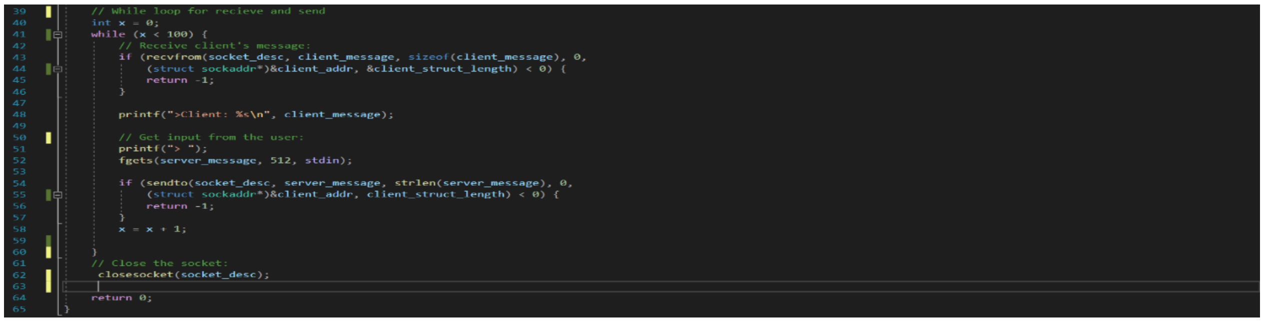


```
    // Create UDP socket:
    socket_desc = socket(AF_INET, SOCK_DGRAM, 0);

    // Set port and IP:
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(2000);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
```



```
    // Bind to the set port and IP:
    if (bind(socket_desc, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        return -1;
    }
    printf("Done with binding\n");
```

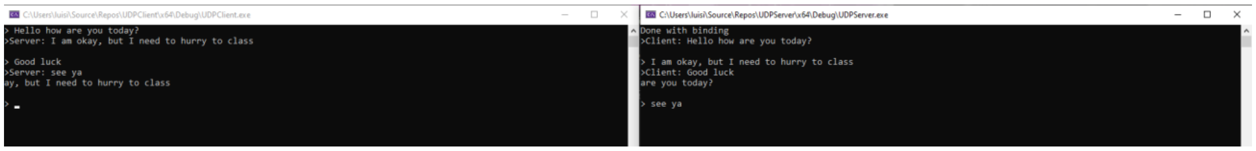
Similar to the TCP Server, the server would wait for a response from the client and display the message with line 48. The server would then be allowed to send a message back to the client, and this process would constantly go on until the client leaves, or they have sent to up to 100 messages.



```
39      // While loop for recieve and send
40      int x = 0;
41      while (x < 100) {
42          // Receive client's message:
43          if (recvfrom(socket_desc, client_message, sizeof(client_message), 0,
44              (struct sockaddr*)&client_addr, &client_struct_length) < 0) {
45              return -1;
46          }
47
48          printf(">Client: %s\n", client_message);
49
50          // Get input from the user:
51          printf("> ");
52          fgets(server_message, 512, stdin);
53
54          if (sendto(socket_desc, server_message, strlen(server_message), 0,
55              (struct sockaddr*)&client_addr, client_struct_length) < 0) {
56              return -1;
57          }
58          x = x + 1;
59
60      }
61      // Close the socket:
62       closesocket(socket_desc);
63
64      return 0;
65  }
```

UDP Client and Server Final Result:

As you can see the back-and-forth messages do work however for some reason in UDP, whenever I would send a message, bits of the previous messages would also be sent. I am not sure if this with my code, or something that UDP naturally causes however the end result does work.



```
C:\Users\luisi\Source\Repos\UDPClient\x64\Debug\UDPClient.exe
> Hello how are you today?
>Server: I am okay, but I need to hurry to class

> Good luck
>Server: see ya
ay, but I need to hurry to class

> 
```

```
C:\Users\luisi\Source\Repos\UDPServer\x64\Debug\UDPServer.exe
Done with binding
>Client: Hello how are you today?

> I am okay, but I need to hurry to class
>Client: Good luck
are you today?

> see ya
```

Note: Complete code is going to be in Zip File.