

# Computer Science Project

**C++ Chat Server and Client**

Made By: Anukool Kesharwani



2017-18

# **INTRODUCTION**

Instant messengers have become a daily part of our life. We cannot imagine a life without Whatsapp, Telegram, Hike, etc. I was always curious about their working, so I spent some time and learnt about their theory of implementation. Equipped with the right knowledge I have created a very basic implementation of instant messengers.

Though this is very basic, probably it might seem like app of MS-DOS era. But the functionality present in it is very similar to the current IM app, it can connect to a client through a server and send messages to other client.

# USER' S GUIDE

Let me first of all explain the contents of the project folder.

Folder **Server**:

- Header Files
- Logs
- Program Code
- Release

Folder **Server**:

- Header Files
- Logs
- Program Code
- Release

In the folder header files, all the custom made header files(.hpp) are

used. In the logs folder all the program run logs are stored, so one can see the logs to find the error behind the crash of the program. In the program code folder, the final .cpp files are stored. the compiled programs are stored in folder release, both 32bit and 64bit versions are stored in it.

Before using the program, I would like to elaborate the conditions under which it can run.

1. Running servers and client, all on the same machine.
2. Running server and clients on different machines on same LAN network.

Please note that this cannot work on PCs connected through the internet, but only on a LAN network.

For case 1: we use the local ip that is 127.0.0.1

For case 2: we use the local ip of the machine on the LAN network, eg:- 192.168.0.89, etc.

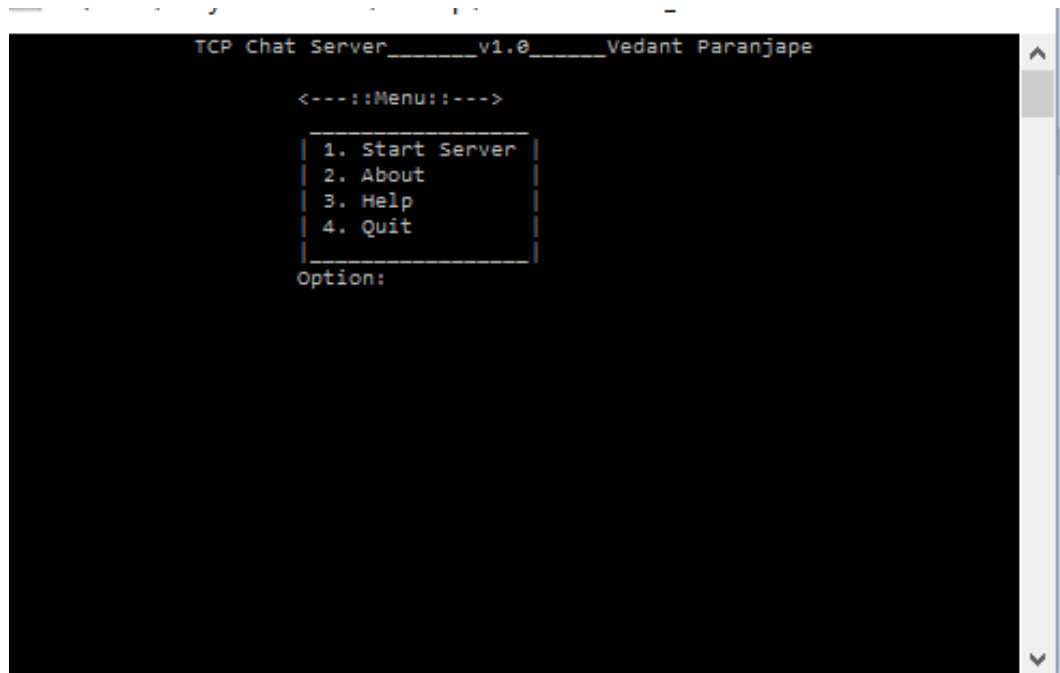
I am running the program on my local machine, so I will set my server's ip address to 127.0.0.1, and connect the client to server at ip 127.0.0.1 . server port used in 444, any available port can be used.

Remember one thing, when one runs the program for the first time on their PC, run the program "RunMeOnlyOnce".

Don't run it again, it is to be run only

once during first use. Please use the 32 bit or 64 bit version, whatever your PC supports.

### Server application:



Now there are 4 options. Type 1 to start server, 2 to show information about the creator of the program that is me. 3 for help and 4 to quit the program.

```
C:\Users\vedant\Documents\Projects\TCP Chat Server>
Made By: Vedant Paranjape
TCP Chat Server
Version: 1.0(alpha)
Made for class XII Computer Science project
Press any key to continue . . . _
```

Output on selecting option 2.

```
C:\Users\vedant\Documents\Projects\TCP Chat Server>
Press 1 to start server
Then enter the ip address on which the server should be accessible...it should be your PC's local ip or 127.0.0.1 for localhost
After that enter the port on which it should work
Now the Server is setup and now clients can connect
Check the documentation for more details...
Press any key to continue . . .
```

Output on selecting option 3.

```
C:\Users\vedant\Documents\Projects\TCP Chat Server>
Enter the server ip address: 127.0.0.1
Enter the server port: 444_
```

output on selecting option 1, i.e. starting the server, there is a prompt to enter the server's ip address and port, through which it can be accessed by the clients. In this case I am running it on 127.0.0.1 , as I have server and client n the same machine.

```

Enter the server ip address: 127.0.0.1

Enter the server port: 444

Intializing Winsock API.....done
Binding socket.....done
Intializing listening socket.....done
Socket intialized 0
Socket intialized 1
Socket intialized 2
Socket intialized 3
Socket intialized 4
Socket intialized 5
Socket intialized 6
Socket intialized 7
Socket intialized 8
Socket intialized 9
Press any key to continue . . . _

```

On pressing enter the server boots up, one can see the progress. Press any key to continue.

```

TCP Chat Server_____v1.0_____Vedant Paranjape

-----
IP Address :127.0.0.1
Port Number:444
-----

```

Now the server is running, one can see the clients connected, their ID's and the clients that have disconnected.

```

-----
TCP Chat Server_____v1.0_____Vedant Paranjape

-----
IP Address :127.0.0.1
Port Number:444
-----

Client connected

```



One can see that a message is displayed when a client is connected, when the client will send the server its ID and the ID of the other client to chat with.

The clients id and the id of client with which it wants to chat is shown as follows:-

```
TCP Chat Server_____v1.0_____Vedant Paranjape
-----
IP Address :127.0.0.1
Port Number:444
-----
Client connected
Client connected
>>ID # 23 sID # 32
>>ID # 32 sID # 23
_
```

when a client disconnects a message is shown as follows:-

```
TCP Chat Server_____v1.0_____Vedant Paranjape
-----
IP Address :127.0.0.1
Port Number:444
-----
Client connected
Client connected
>>ID # 23 sID # 32
>>ID # 32 sID # 23
Client 32 has disconnected
Client 23 has disconnected
```

To shutdown the server, type "quit" directly in the terminal, and press enter.

```
TCP Chat Server_____v1.0_____Vedant Paranjape
-----
IP Address :127.0.0.1
Port Number:444
-----
client connected
client connected
>>ID # 23 sID # 32
>>ID # 32 sID # 23
client 32 has disconnected
client 23 has disconnected
quit
```

### Client Application:

For this to work, first of all the server must be running, start the server, following the instructions above.

```
Chat Client v1.0_Alpha <====> Client
Enter the ip address of the server: 127.0.0.1
Enter the port of the server: 444
```

Enter the ip address of the chat server and the port on which it is accessible, and then press enter. After that you will get the prompt to enter your ID and the ID with whom you want to chat.

Please note that the ID selected must be a number between 10-99, selecting any other number will result in undefined behaviour, and also make sure that the same ID is not used by any other client. I used 12 as my ID and I want to chat with client having ID 45.

Use any number between 10-99 and then enter it and press enter.

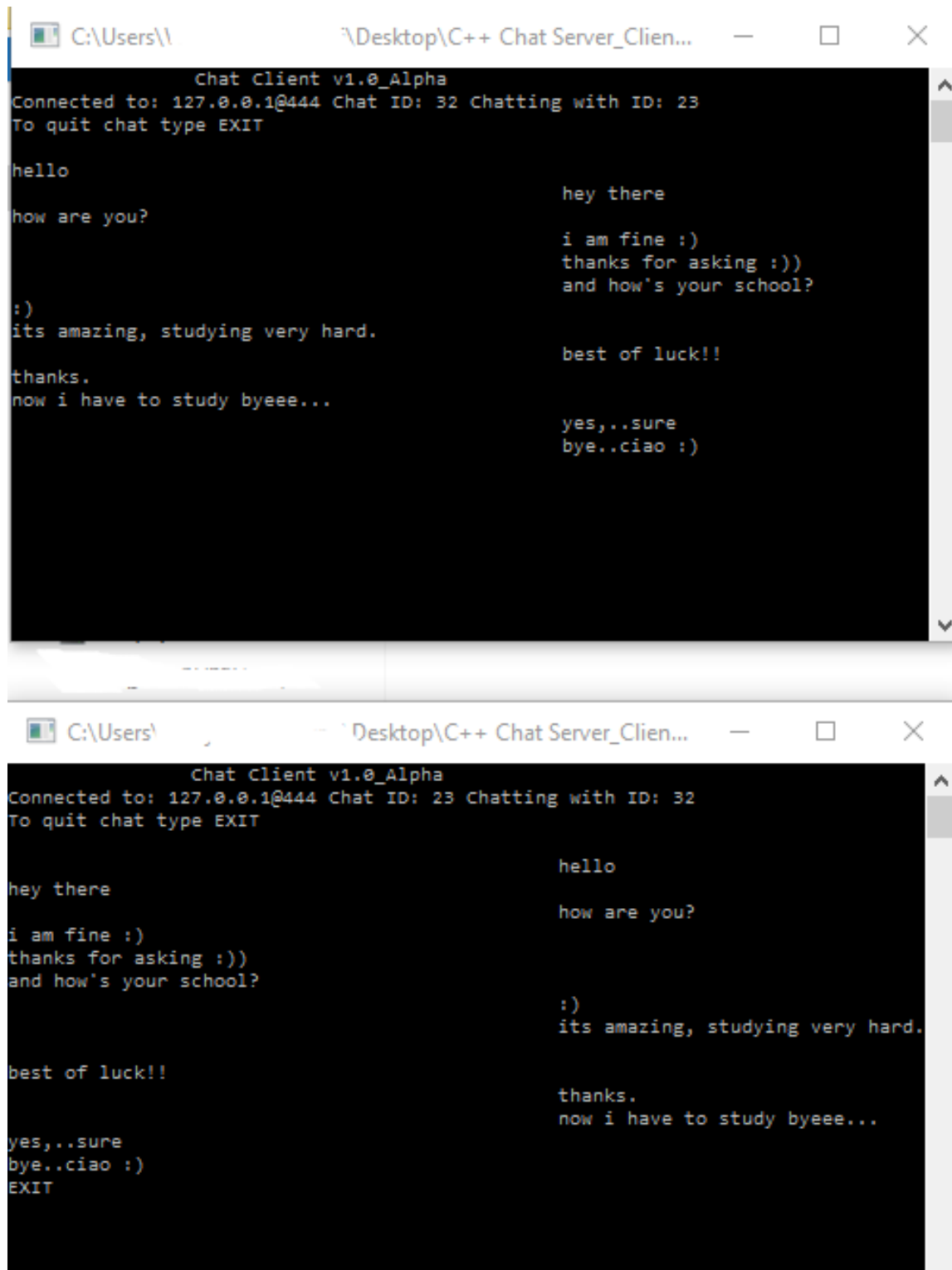
```
Chat Client v1.0_Alpha <====> Client
Enter the ip address of the server: 127.0.0.1
Enter the port of the server: 444
Initializing Winsock API.....done
Socket Initialised
Connected to server
Command: Enter your id and id of client with whom you want to chat(the id can be from 10-99 only)
Enter your ID: 12
Enter the ID with whom you want to chat: 45
```

Now the client is setup and you can simply chat with another client, your messages are shown on left side and the other clients messages are shown on right side. If you want to stop the chat and exit type "EXIT".

```
Chat Client v1.0_Alpha
Connected to: 127.0.0.1@444 Chat ID: 12 Chatting with ID: 45
To quit chat type EXIT

EXIT_
```

## Server and client working



```
Chat Client v1.0_Alpha
Connected to: 127.0.0.1@444 Chat ID: 32 Chatting with ID: 23
To quit chat type EXIT

hello
how are you?
:)
its amazing, studying very hard.
thanks.
now i have to study byeee...

hey there
i am fine :)
thanks for asking :))
and how's your school?
best of luck!!
yes,..sure
bye..ciao :)

Chat Client v1.0_Alpha
Connected to: 127.0.0.1@444 Chat ID: 23 Chatting with ID: 32
To quit chat type EXIT

hey there
i am fine :)
thanks for asking :))
and how's your school?
best of luck!!
yes,..sure
bye..ciao :)
EXIT
```

```
TCP Chat Server_____v1.0_____Vedant Paranjape

-----
IP Address :127.0.0.1
Port Number:444
-----

Client connected
Client connected
>>ID # 23 sID # 32
>>ID # 32 sID # 23
```

The first image is of the first client, second image is of the second client and the third image is of the server. As you can see one can connect and chat as one wishes, just like Whatsapp, but in retro style.

# SOURCE CODE

## Client header files

### 1. Class\_ClientNetworkInit.hpp

```
#include <iostream>
#include <winsock2.h>
#include <windows.h>
#include "Class_Logger.hpp"
using namespace std;

SOCKET ServerCom = INVALID_SOCKET;
int ClientAddressSize;
SOCKADDR_IN ClientAddress;
char lftxt[40] = "../Logs/ClientNetworkInitLogs";
logfile lf(lftxt);

class ClientNetworkInit
{
private:
    WSADATA ClientNetworkData;
    char message[512];

public:
    ClientNetworkInit(char sIP[],u_short sPORT)
    {
        lf.writeLog(sIP);
        cout<<"Intializing Winsock API";
        strcpy(lftxt,"Initializing Winsock API");
        lf.writeLog(lftxt);

        for(int i=0;i<15;i++)
        {
            cout<<".";
            Sleep(100);
        }

        if(!WSAStartup(MAKEWORD(2,1),(WSADATA*)&ClientNetworkData))
        {
            cout<<"done"<<endl;
            strcpy(lftxt,"done");
            lf.writeLog(lftxt);
        }
        else
        {
            cout<<"Error starting Winsock API"<<endl;
            strcpy(lftxt,"Error starting Winsock API");
            lf.writeLog(lftxt);

            cout<<"Shutting down the server"<<endl;
            strcpy(lftxt,"Error starting Winsock API");
            lf.writeLog(lftxt);
            lf.~logfile();
        }
    }
};
```

```

        closesocket(ServerCom);
        WSACleanup();
        exit(0);
    }

    ClientAddress.sin_addr.s_addr = inet_addr(sIP);
    ClientAddress.sin_port = htons(sPORT);
    ClientAddress.sin_family = AF_INET;

    ClientAddressSize = sizeof(ClientAddress);
    ServerCom = socket(AF_INET, SOCK_STREAM, 0);

    if(ServerCom == INVALID_SOCKET)
    {
        cout<<"Error initialising socket"<<endl;
        strcpy(lftxt, "Error starting socket");
        lf.writeLog(lftxt);
        lf.~logfile();
        closesocket(ServerCom);
        WSACleanup();
        exit(0);
    }
    else
    {
        cout<<"Socket Intialised"<<endl;
        strcpy(lftxt, "Socket Intialised");
        lf.writeLog(lftxt);
    }
}

void ConnectServer()
{
    int val;
    val =
connect(ServerCom, (SOCKADDR*)&ClientAddress, ClientAddressSize);
    if(!val)
    {
        cout<<"Connected to server"<<endl;
        strcpy(lftxt, "Connected to server");
        lf.writeLog(lftxt);
    }

    else
    {
        cout<<"Unable to connect to the server"<<endl;
        strcpy(lftxt, "Unable to connect to the server");
        lf.writeLog(lftxt);

        cout<<"Closing chat client"<<endl;
        strcpy(lftxt, "Closing chat client");
        lf.writeLog(lftxt);
        lf.~logfile();

        WSACleanup();
        exit(0);
    }
}

};

```

## 2. Class\_Logger.hpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <ctime>
#include <algorithm>
#include <stdlib.h>
#include "Function_RunCounter.hpp"
using namespace std;

class logfile
{
    private:
        time_t now;
        char* time_now;
        fstream WriteLog;
    public:
        logfile(char filename[])
        {
            char count[2];
            snprintf(count, sizeof(count), "%d", ReadRunCounter());
            now = time(0);
            time_now = ctime(&now);

            WriteLog.open(strcat(strcat(filename, count), ".txt"), ios::out);
            WriteLog<<"Run Time : "<<time_now;
            WriteLog<<endl;
        }

        void writeLog(char event[])
        {
            WriteLog<<event;
            WriteLog<<endl;
        }

        ~logfile()
        {
            WriteLog.close();
        }
};
```

## 3. Function\_ClientNetworkCom.hpp

```
#include <iostream>
#include <string>
#include "Function_GetAuthenticationData.hpp"
using namespace std;

struct chatData
{
    int myid;
    int chatid;
};
```



```

void DataExt(char msg[], chatData& data)
{
    char tmp[3];
    char tmp0[3];

    for(int i=0; i<2; i++)
    {
        tmp[i] = msg[i];
    }
    for(int i=0; i<2; i++)
    {
        tmp0[i] = msg[i+2];
    }

    data.myid = atoi(tmp);
    data.chatid = atoi(tmp0);
}

void ClientNetworkComInit(char ip[], u_short port)
{
    string cMessageRecv;
    char cMessage[512] = " ";
    int cstate = 0;
    chatData data;

    do
    {
        cstate = recv(ServerCom, cMessage, 512, 0);
    } while(cstate <= 0);
    cMessageRecv = cMessage;
    cout<<"Command: "<<cMessageRecv<<" (the id can be from 10-99
only)"<<endl;
    cstate = 0;
    strcpy(cMessage, " ");

    GetAuthenticationData(cMessage);
    cin.ignore();

    do
    {
        cstate = send(ServerCom, cMessage, 512, 0);
    } while(cstate <= 0);
    cstate = 0;

    DataExt(cMessage, data);

    system("cls");

    cout<<"\t\tChat Client v1.0_Alpha"<<endl;
    cout<<"Connected to: "<<ip<<"@"<<port<<" Chat ID: "<<data.myid<<"
Chatting with ID: "<<data.chatid<<endl;
    cout<<"To quit chat type EXIT"<<endl<<endl;
}

//call this in a thread
void ClientNetworkComSend()
{
    char MessageSend[512] = " ";
    int mstate = 0;

    while(strcmp(MessageSend, "EXIT"))

```

```

        {
            cin.getline(MessageSend, 512);
            do
            {
                mstate = send(ServerCom, MessageSend, 512, 0);
            } while (mstate <= 0);
        }
        exit(0);
    }

// call this is separate thread
void ClientNetworkComRecv()
{
    string cMessageRecv;
    char cMessageRcv[512] = "";
    int flag = 0;

    while (strcmp(cMessageRcv, "EXIT"))
    {
        do
        {
            flag = recv(ServerCom, cMessageRcv, 512, 0);
        } while (flag <= 0);

        cMessageRecv = cMessageRcv;
        cout << "\t\t\t\t\t\t\t" << cMessageRecv << endl;
    }
    exit(0);
}

```

## 4. Function\_GetAuthenticationData .hpp

```

#include <iostream>
#include <string>
using namespace std;

void GetAuthenticationData(char msg[]);

void GetAuthenticationData(char msg[])
{
    string myid;
    string chatid;

    cout << "Enter your ID: ";
    cin >> myid;

    cout << "Enter the ID with whom you want to chat: ";
    cin >> chatid;

    myid = myid + chatid;

    strcpy(msg, myid.c_str());
}

```

## 5. Function\_RunCounter.hpp

```
#include <iostream>
#include <fstream>
using namespace std;

int ReadRunCounter();
void WriteRunCounter();

int ReadRunCounter()
{
    int counter;
    fstream file;
    file.open("../Logs/RunCounter.txt",ios::in);

    file>>counter;
    file.close();
    return counter;
}

void WriteRunCounter()
{
    int tempcount = ReadRunCounter();
    tempcount++;

    fstream fileWrite;
    fileWrite.open("../Logs/RunCounter.txt",ios::out);

    fileWrite<<tempcount;
    fileWrite.close();
}
```

## Client Program Codes

### 1. ChatClient.cpp

```
#include <iostream>
#include <thread>
#include "../Header files/Class_ClientNetworkInit.hpp"
#include "../Header files/Function_ClientNetworkCom.hpp"
using namespace std;

main()
{
    WriteRunCounter(); //updates the counter which counts
the number of times this program is run
    char ip[15]; //variable to store the ip address
    u_short port;
    thread recv;

    cout<<"\t\tChat Client v1.0_Alpha <====> Client"<<endl;

    cout<<"Enter the ip address of the server: ";
    cin>>ip;
```

```

        cout<<"Enter the port of the server: ";
        cin>>port;

        ClientNetworkInit cli(ip,port);           //construct the
ClientNetworkInit object
        cli.ConnectServer();                     //connect to the specified
server

        ClientNetworkComInit(ip,port);           //setup to enter your id and
the id of the user you want to chat with

        recv = thread(ClientNetworkComRecv);      //start the reciever on
a separate thread
        recv.detach();

        ClientNetworkComSend();                  //run the sender in the main
thread
    }

```

## 2. RunMeOnlyOnce.cpp

```

#include <iostream>
#include <fstream>
using namespace std;

main()
{
    fstream file;
    file.open("../Logs/RunCounter.txt",ios::out);
    int val = 0 ;

    file<<val;

    file.close();
    cout<<"Counter set, dont run this again"<<endl;
    exit(0);
}

```

## Server header files

### 1. Class\_Logger.hpp

```

#include <iostream>
#include <fstream>
#include <string.h>
#include <ctime>
#include <stdlib.h>
#include "Function_RunCounter.hpp"
using namespace std;

class logfile
{
    private:
        time_t now;           //required variables to get the
current time
        char* time_now;
        fstream WriteLog;     //declaration of fstream object to
write logs to a file
    public:
        logfile(char filename[]) //class constructors

```

```

        {
            char count[3];
            sprintf(count, sizeof(count), "%d", ReadRunCounter());
//convert runcount to char
            now = time(0);
            time_now = ctime(&now);

            WriteLog.open(strcat(strcat(filename, count), ".txt"), ios::out);
//open file to write log
            WriteLog<<"Run Time : "<<time_now; // write to file
            WriteLog<<endl;
        }

        void writeLog(char event[]) //function to write event logs
to the file
        {
            WriteLog<<event;
            WriteLog<<endl;
        }

        ~logfile() //desctructor
        {
            WriteLog.close();
        }
};

```

## 2. Class\_NetworkInit.hpp

```

#include <iostream>
#include <winsock2.h>
#include <windows.h>
#include <process.h>
#include <thread>
#include <fstream>
#include <string>
#include <memory>
#include "Function_NetworkCom.hpp"
using namespace std;

//////////////////// Variables //////////////////////////////////
int ClientStore = 0; //
char lftxtnio[40] = "../Logs/NetworkInitLogs"; //
char lftxtni[50]; //
logfile lfni(lftxtnio); //
////////////////////////////////////

//////////////////// Class NetworkInit //////////////////////////////////
////////////////////////////////////
class NetworkInit
{
private:
    //declarations of winsock library objects necessary for the
sockets parts to run
    WSADATA NetworkData;
    SOCKET ListenSocket;
    SOCKADDR_IN Address;
    int AddressSize;

public:
    NetworkInit(); // default constructor

```

```

NetworkInit(char IP[],u_short PORT) // argumented constructor
{
    lfni.writeLog(IP); // write server ip to log file
//-----Initailize Winsock API-----
-----
//process to start the winsock api

strcpy(lftxtni,"Intializing Winsock API");
lfni.writeLog(lftxtni);
cout<<"Intializing Winsock API";
for(int i=0;i<17;i++)
{
    cout<<".";
    Sleep(100);
}
if(!WSAStartup(MAKEWORD(2,1),(WSAData*)&NetworkData))
{
    cout<<"done"<<endl;
    strcpy(lftxtni,"done");
    lfni.writeLog(lftxtni);
}
else
{
    cout<<"Error starting Winsock API"<<endl;
    strcpy(lftxtni,"Error starting Winsock API");
    lfni.writeLog(lftxtni);

    cout<<"Shutting down the server"<<endl;
    strcpy(lftxtni,"Shutting down the server");
    lfni.writeLog(lftxtni);
    lfni.~logfile();

    closesocket(ListenSocket);
    for(int i=0;i<10;i++)
    {
        closesocket(ClientConn[i]);
    }
    WSACleanup();
    exit(0);
}
//-----
-----

Address.sin_addr.s_addr = inet_addr(IP);
Address.sin_port = htons(PORT);
Address.sin_family = AF_INET;

//-----Binding Listening Socket-----
-----
//process of binding the socket

cout<<"Binding socket";
strcpy(lftxtni,"Binding socket");
lfni.writeLog(lftxtni);
for(int i=0;i<26;i++)
{
    cout<<".";

```

```

        Sleep(50);
    }
    ListenSocket = socket(AF_INET, SOCK_STREAM, 0);
    if(!bind(ListenSocket, (SOCKADDR*)&Address, sizeof(Address)))
    {
        cout<<"done"<<endl;
        strcpy(lftxtni, "done");
        lfni.writeLog(lftxtni);
    }
    else
    {
        cout<<"Error Binding the port"<<endl;
        strcpy(lftxtni, "Error Binding the port");
        lfni.writeLog(lftxtni);

        cout<<"Shutting down the server"<<endl;
        strcpy(lftxtni, "Shutting down the server");
        lfni.writeLog(lftxtni); // write to log file
        lfni.~logfile();

        closesocket(ListenSocket);
        for(int i=0; i<10; i++)
        {
            closesocket(ClientConn[i]);
        }
        WSACleanup();
        exit(0);
    }
}

//-----
//-----
//-----

//-----Initialize listening socket-----
//-----

//initialize the listening socket

cout<<"Initializing listening socket";
strcpy(lftxtni, "Initalizing listening socket");
lfni.writeLog(lftxtni);

for(int i=0; i<11; i++)
{
    cout<<".";
    Sleep(100);
}
if(!listen(ListenSocket, SOMAXCONN))
{
    cout<<"done"<<endl;
    strcpy(lftxtni, "done");
    lfni.writeLog(lftxtni);
}
else
{
    cout<<"Error in starting listening socket"<<endl;
    strcpy(lftxtni, "Error in starting listening socket");
    lfni.writeLog(lftxtni);

    cout<<"Shutting down the server"<<endl;
    strcpy(lftxtni, "Shutting down the server");
}

```

```

        lfni.writeLog(lftxtni);
        lfni.~logfile();
        closesocket(ListenSocket);
        for(int i=0;i<10;i++)
        {
            closesocket(ClientConn[i]);
        }
        WSACleanup();
        exit(0);
    }
//-----
-----

AddressSize = sizeof(Address);
for(int i=0;i<10;i++)
{
    ClientConn[i] = INVALID_SOCKET;
    ClientConn[i] = socket(AF_INET, SOCK_STREAM, 0);

    if(ClientConn[i] == INVALID_SOCKET)
    {
        cout<<"Error in initialising listening socket
"<<i<<endl;
        strcpy(lftxtni, "Error in initialising listening
socket ");
        lfni.writeLog(lftxtni);
        lfni.~logfile();
    }
    else
    {
        cout<<"Socket intialized "<<i<<endl;
    }
    Sleep(200);
}

system("pause");
system("cls");

cout<<"\t\tTCP Chat Server_____v1.0_____Vedant
Paranjape"<<endl<<endl; // display of server information
cout<<"\t\t-----"<<endl;

cout<<"\t\t IP Address : "<<IP<<endl;
cout<<"\t\t Port Number: "<<PORT<<endl;
cout<<"\t\t-----"<<endl;
}
//-----Intitalize all client
handler sockets-----
-----

void AcceptConn()
{
    for(;;)
    {
        if((ClientConn[ClientStore] =
accept(ListenSocket, (SOCKADDR*)&Address, &AddressSize))
        {
            cout<<"Client connected "<<endl;

```



```

        send(ClientConn[ClientStore],"Enter your id
and id of client with whom you want to chat",57,0); // send necessary
instructions to the client
        CreateClientThread(ClientStore); // call
function to create new client managing thread
        ClientStore++; //increment
the client counter
    }
}

~NetworkInit()
{
    // destructor -- shutdown the winsock api, delete the socket
object
    for(int i=0;i<10;i++)
    {
        cout<<"Closing socket "<<i<<endl;
        closesocket(ClientConn[i]);
    }
    cout<<"Shutting down Winsock API"<<endl;
    strcpy(lftxtni,"Shutting down Winsock API");
    lfni.writeLog(lftxtni);
    lfni.~logfile();

    closesocket(ListenSocket);
    WSACleanup();
    exit(0);
}
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

### 3. Function\_Menu.hpp

```

#include <iostream>
#include <process.h>
using namespace std;

int Menu();
int Menu()
{
    int option; //define option variable
    system("cls"); //clear the screen
    cout<<"\t\tTCP Chat Server_____v1.0_____Vedant
Paranjape"<<endl<<endl;
    cout<<"\t\t\t<---:Menu::--->"<<endl;
    cout<<"\t\t\t_____ "<<endl;
    cout<<"\t\t\t| 1. Start Server | "<<endl;
    cout<<"\t\t\t| 2. About | "<<endl;
    cout<<"\t\t\t| 3. Help | "<<endl;
    cout<<"\t\t\t| 4. Quit | "<<endl;
    cout<<"\t\t\t|_____ "<<endl;
    cout<<"\t\t\tOption: ";
    cin>>option; //take the input of option
    cin.ignore();

    return option;
}

```

## 4. Function\_MessageParser

```
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;

struct AuthDataParsed
{
    int ID;
    int sID;
};

void AuthDataParser(char RawMessage[],AuthDataParsed& msg);

void AuthDataParser(char RawMessage[],AuthDataParsed& msg)
{
    char id[3] = "";
    char sid[3] = "";

    for(int i=0;i<2;i++)
    {
        id[i] = RawMessage[i];
    }

    for(int i=0;i<2;i++)
    {
        sid[i] = RawMessage[i+2];
    }

    msg.ID = atoi(id);
    msg.sID = atoi(sid);
}
```

## 5. Function\_NetworkCom.hpp

```
#include <iostream>
#include <string>
#include <stdlib.h>
#include <thread>
#include <winsock2.h>
#include <windows.h>
#include "Library_IDManager.hpp"
#include "Function_MessageParser.hpp"
#include "Class_Logger.hpp"
using namespace std;

////////// variables //////////
thread HandleClient[10];
SOCKET ClientConn[10];
char lftxt[20] = "../Logs/ChatLogs";
logfile lf(lftxt);
////////////////////////////////////

//////////////////////////////////// Function NetworkCom
////////////////////////////////////
void NetworkCom(int loc);

void NetworkCom(int loc)
```

```

{
    int flag = 0;
    char datatemp[512] = " ";
    char msgtemp[512] = " ";
    AuthDataParsed AuthData;

    do
    {
        flag = recv(ClientConn[loc], datatemp, 512, 0);
    } while (flag <= 0);
    flag = 0;

    AuthDataParser(datatemp, AuthData);
    IDStore(AuthData.ID, loc);

    cout<<">>"<<"ID # "<<AuthData.ID<<" sID # "<<AuthData.sID<<endl;

    while(ClientConn[loc] != INVALID_SOCKET)
    {
        flag = recv(ClientConn[loc], msgtemp, 512, 0);
        if(flag > 0)
        {
            lf.writeLog(datatemp);
            lf.writeLog(msgtemp);
        }
        flag = 0;

        flag =
send(ClientConn[ReturnLocation(AuthData.sID)], msgtemp, 512, 0);
        if(flag > 0)
        {
            lf.writeLog(datatemp);
            lf.writeLog(msgtemp);
        }
        flag = 0;

        if(!strcmp(msgtemp, "EXIT"))
        {
            closesocket(ClientConn[loc]);
            cout<<"Client "<<AuthData.ID<<" has
disconnected"<<endl;
            cout<<"Client "<<AuthData.sID<<" has
disconnected"<<endl;
            break;
        }
    }

}

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////

//////////////////////////////////// Function
CreateClientThread
////////////////////////////////////
////////////////////////////////////
void CreateClientThread(int location);

void CreateClientThread(int location)
{
    HandleClient[location] = thread(NetworkCom, location);
}

```

```

        HandleClient[location].detach();
    }
    //////////////////////////////////////
    //////////////////////////////////////
    //////////////////////////////////////

```

## 6. Function\_RunCounter.hpp

```

#include <iostream>
#include <fstream>
using namespace std;

int ReadRunCounter();
void WriteRunCounter();

int ReadRunCounter()
{
    int counter;
    fstream file;
    file.open("../Logs/RunCounter.txt",ios::in);

    file>>counter;
    file.close();
    return counter;
}

void WriteRunCounter()
{
    int tempcount = ReadRunCounter();
    tempcount++;

    fstream fileWrite;
    fileWrite.open("../Logs/RunCounter.txt",ios::out);

    fileWrite<<tempcount;

    fileWrite.close();
}

```

## 7. Function\_RunCounter.hpp

```

#include <iostream>
using namespace std;

////////////////////////////////////
int ID_Array[10]; //
////////////////////////////////////

////////////////////////////////////
void IDStore(int ID,int ArrayLoc); //
void IDStore(int ID,int ArrayLoc) //
{
    ID_Array[ArrayLoc] = ID; //
}
////////////////////////////////////

////////////////////////////////////
int IDReturn(int ArrayLoc); //
int IDReturn(int ArrayLoc) //
{
    //

```

```

        return ID_Array[ArrayLoc];    //
    }                                  //
    ////////////////////////////////////
    ////////////////////////////////////
int ReturnLocation(int ID);           //
int ReturnLocation(int ID)           //
{                                     //
    int lo = 0;                       //
                                     //
    for(int i=0;i<10;i++)             //
    {                                  //
        if(ID_Array[i] == ID)         //
        {                             //
            lo = i;                   //
        }                             //
    }                                  //
    return lo;                         //
}                                     //
    ////////////////////////////////////

```

## Server Program Codes

### 1. ChatServer.hpp

```

#include <iostream>
#include <process.h>
#include <conio.h>
#include "../Header Files/Function_Menu.hpp"
#include "../Header Files/Class_NetworkInit.hpp"
using namespace std;

void CommandHandle(NetworkInit& a);

void CommandHandle(NetworkInit& a)
{
    char command[5];

    cin.ignore();

    while(1)
    {
        if(kbhit())
        {
            cin.getline(command,20);

            if(!strcmp(command,"quit"))
            {
                a.~NetworkInit();
                break;
            }
        }
    }
}

void RunServer();

void RunServer()

```

```

{
    WriteRunCounter();
    char IP[16];
    u_short PORT;
    thread CommandHandler;

    system("cls");
    cout<<"\n\n\t\tEnter the server ip address: ";
    cin>>IP;
    cout<<endl;

    cout<<"\t\tEnter the server port: ";
    cin>>PORT;
    cout<<endl;

    NetworkInit com(IP,PORT);

    CommandHandler = thread(CommandHandle,ref(com));
    com.AcceptConn();
}

void about();

void about()
{
    system("cls");
    cout<<"Made By: Vedant Paranjape"<<endl;
    cout<<"TCP Chat Server"<<endl;
    cout<<"Version: 1.0(alpha)"<<endl;
    cout<<"Made for class XII Computer Science project"<<endl;
    system("pause");
}

void help();

void help()
{
    system("cls");
    cout<<"Press 1 to start server"<<endl;
    cout<<"Then enter the ip address on which the server should be  
accessible...it should be your PC's local ip or 127.0.0.1 for  
localhost"<<endl;
    cout<<"After that enter the port on which it should work"<<endl;
    cout<<"Now the Server is setup and now clients can connect"<<endl;
    cout<<"Check the documentation for more details..."<<endl;
    system("pause");
}

main()
{
    do
    {
        switch(Menu())
        {
            case 1:
                RunServer();
                break;

            case 2:
                about();
                break;
        }
    }
}

```

```

        case 3:
            help();
            break;

        case 4:
            exit(0);
            break;

    }
}while(1);
}

```

## 2. RunMeOnlyOnce.hpp

```

#include <iostream>
#include <fstream>
using namespace std;

main()
{
    fstream file;
    file.open("../Logs/RunCounter.txt",ios::out);
    int val = 0 ;

    file<<val;

    file.close();
    delete &val;
    delete &file;
    exit(0);
}

```