Server

structure Client
{
    struct sockaddr_in address;
    int sockfd;
    int uid;
    char name;
    char pass;
}

- main() takes port number as CML argument
    - If argc != 2, return failure
    - Socket creation
    - Building socket structure
    - Bind
    - Listen to connections

    while(true):
        Accept connections and keep server waiting for new clients
        Adding clients to queue : queue_add()
        Create threads for each client
        Sleep for 1 sec, reducing CPU uses.

- queue_add()
    - Use mutex lock for retaining each client's information
    - Check if max no. of Clients reached

      If true:
        Display address and send rejection message to client
      Else:
        The user is added to the active users array

- handle_client()
    - Check whether the received name is within 2 to 32 characters.
        - if true, copy client's name client structure using client's uid
        - else, Error : ask to rejoin

    - Keep a variable to count no of initial communications between client and server, So rest of the information about client can be stored.

      Variable count =0

- while(true):
    - int receive = Call recv function to get message from client
    - Break out of loop if leave_flag =1 (this variable checks for error and to break out of while loop)
    - if(receive>0) :
        if(count ==0) : upto this only client name is taken
            Ask for clients pass ,store into clients structure
            Ask if client is already registered or new

            if(client is new):
                Store clients name and pass in txt file and allow chatting.
                    if(registered client):
                        Verify credentials from passfile.txt
                        If verified : allow chatting
                        Else: send rejection message and destroy the thread connecting this client to server

            Increase count to come out of first if condition, verifying clients credentials.

    - if(receive >0 && count >1)
        Connection established and now call send_message() ,function to send one client message to all active users through the server.
        Store all communication between clients into logfile.txt

    - if(receive ==0 || client hits “exit”)
        Come out of while loop
    - else(check for error) : to come out of loop

- After coming out of while loop :
    - Remove client from active users list
    - Destroy client’s thread

➢ send_individual_client_message(string, cli –> uid)
- Send message to only client whose uid matches.

➢ send_message(string, cli –> uid)
- Use mutex lock for each client’s message so that the message remains unaffected
- Send message to all active clients except the one who send it.

- str_trim_lf()
    - Trimming (\n is replaced with \0)

- queue_remove()
    - Use mutex lock for removing a client while retaining existing users' information
    - The user is removed from the active user list

Client

- main() takes port number as CML argument
    - If argc != 2, return failure
    - Enter user name and trim it
    - Check whether the received name is within 2 to 32 characters.
        - if false, return failure
    - Socket creation
    - Building socket structure
    - Connecting to the server
    - Send name to the server for validation
    - Creating thread to send message
    - Creating thread to receive message
    - Press ctrl+c or "exit" to close client thread
    - Close the socket
- send_msg_handler()
    - Take message as input from user and trim it
    - Check the message
        - If "exit", exit from chat application
        - Else, send the message to server as well as other active users
    - Press ctrl+c to exit
- recv_msg_handler()
    - If message is received, print it on the user console

Search chat history

- main()
    - Open the file containing chat history
    - Enter the keyword to be searched
        - If found, print the whole line along with the user's name who wrote it
        - Else, print not found