This specification is a tailored specification built around the implementation of RFC 1459 submitted herein, known as networkingirc built by Julian Lazaras.

underlying protocol utilizations:
this implementation of the IRC protocol, or the miniature version of it that it is, is built on top of standard transmission control protocol over unencrypted sockets using a client/server architecture in which clients forward messages through servers which maintain a globalized state for each channel, and each use registered to the server.

a note about concurrency:
the current implementation uses tokio's async/await framework wherein a thread on the server can lay dormant waiting for client communications to trigger an necessary data exchange, such as loading messages only when a join is issued thereby taking up less clock cycles, the implementation herein also takes advantage of syncronization primitives in favor of a database engine, in order to keep the implementation light, and highly parallel. as such each server's entire message history is stored within a single complex hash map. while this approach is viable for small implementations, any large scale chat application should switch to the use of a database engine such as mongodb, sqlite3, or postgres as they are similarly open and free use, but are better optimized for larger data operations than can be stored in a hash map.

the primary means of control and communication established within this protocol are that of the commands, which are defined below. each command is an upper case sequence of ascii compatible characters followed by a space separated list of arguments, and terminated with carriage-return and line feed or \r\n (in theory) in practice this implementation relies on discretized TCP/IP read/writes to distinguish individual commands, and doesn't send bulk commands (were it that I had more time, I probably would adhere more precisely to IRC specifications, but in the interest of developing a minimum viable product, some shortcuts were taken)

the general command structure is

:<prefix> command arg1, arg2, arg3 :<trailing_commets>
command arg1, arg2, arg3, …arg15

if no recognized command is specified it is assumed to be a public message, and is treated as such.

The primary commands implemented in this RC 1459 mock minimization are:

```
          "JOIN"
arguments: <channellist>
channellist is a required argument, but may contain only one channel, the
last channel in the list will become the active channel, and have its
messages loaded. If the room/channel doesn't exist when this command is
```

processed by the server, the chanel will be created according to RFC 1459, however in addition to the creation of the room/channel a default message will be appended to the rooms message board announcing the creation of a new room.

"LIST"

arguments: <channellist>

The list command serves two purposes, listing the channels on the server, and listing the topics of each channel specified in the channel list field

"PART"

arguments: <channellist>

the channel list argument is required as no default behavior is implemented if no rooms/channels are specified. However when specified the list contained within <channellist> are the channels that the user will be removed from.

"QUIT"

arguments: none

server cleans up the thread that it maintains for interacting with this client, and this client exits returning a zero for exit status.

"USER"

arguments: <username> <hostname> <servername> <realname>

all of these arguments are required

with username being a compound unique key with hostname, servername, and real name to derive a unsigned 64 bit user identifier within this implementation.

This command is used to register a user upon connection to a server, and is required in order for a connection to be properly established.

"PING"

used to ensure the authenticity of a client, this is done to ensure the client is reachable, and running the same, or a similar protocol, the current implementation has no arguments, however a proper implementation would have a unique identifier sent to the client. PLEASE NOTE: the implementation does not have the server currently able to respond to PING commands, only the client can do that.

"PONG"

pong is the response to ping, and operates in much the same manner, similarly in this implementation no arguments are required.

"NAMES"

arguments: <channellist>

the default action without an argument is to list all users on the server, this is currently the only feature of this command that is fully

```
implemented, specifying a channellist will cause this command to do
nothing.
```

the other main requirements for operating this implementation of the protocol are the command line arguments, which specify presents as to avoid the need to enter them into a terminal user interface, these arguments, while documented in the code itself, are as follows:

-a, --address the ip address and port on which to run the server, or that the client should connect to
-u, --username the username registered with the server
-h, --hostname the hostname registered with the server, future implementations may grab this from /etc/hostname or similar, however this is the current approach to gaining this information
-r, --realname the real name of the user such as “Julian Lazaras”
-n, --nickname while this is part of the argument parsing system, it doesn’t functional do anything at the moment, as nicknames where a planned, and nearly implemented feature, that didn’t get implemented due to time constraints.

difficulties in developing this specification:
it was tricky to determine the exact structure of IRC messages, principally finding confirmation that a blank message with no command was indeed a public message turned out to be fairly challenging. In addition the determination that all prefixes could be denoted with : as an identification that it was in fact a prefix was challenging given the unfamiliar syntax in RFC 1459 which covered this precise syntax of the protocol (this has been a learning experience in reading docs as well)

How to compile:
compile with cargo:
cargo build --release
cargo build

will built all the necessary code to run this program, to run this project’s code use:
./target/<debug|release>/server --address 127.0.0.1:2323
for the server
and
./target/<debug|release>/client --address 127.0.0.1:2323 --hostname hephaestus --realname "Julian Lazaras" --username cardinal
to start up a client