ECE470:Network Client-Sever Programming

Project 1: Part 1

Aaron Jesus Valdes

March 2, 2021

Contents

1 Project Description

Designing a smart home remote access system which follows the Network/Client architecture. This system allows the user to control certain devices from their home such lights, alarm, and locks from a remote location.

2 Specifications

- Supports smart lights
  - Can turn on and off lights
  - Can change the color and brightness of lights
  - Groups lights into rooms
- Supports alarm system
  - Can arm and disarm the alarm
  - Can have more than one alarm system
  - Can store 4 digit pin per alarm
- Supports Electronic Locks
  - Can open and lock the locks
  - Can have more than 4 locks
  - Can store a 5 digit pin per lock
- Supports more devices
  - Can control the thermostat
    - Turn on or off heat/cold
    - Set temperature
  - Can check the security cameras
    - Can store the video from a security camera
  - Can control blinds and drapes
    - Opens and closes the blinds

3 Assumptions

- We are only working with a single house
- There are multiple rooms in the house
- Each room has multiple lights

- There is only one alarm
- There are at least 4 locks
- One user at a time can connect
- The server is in the home
- The server can be accessed both inside and outside of the house
- The user needs to check and alter status of the devices
- The method of communication is
- The server must be protected by having a login page

4 Initial Design

Data Model

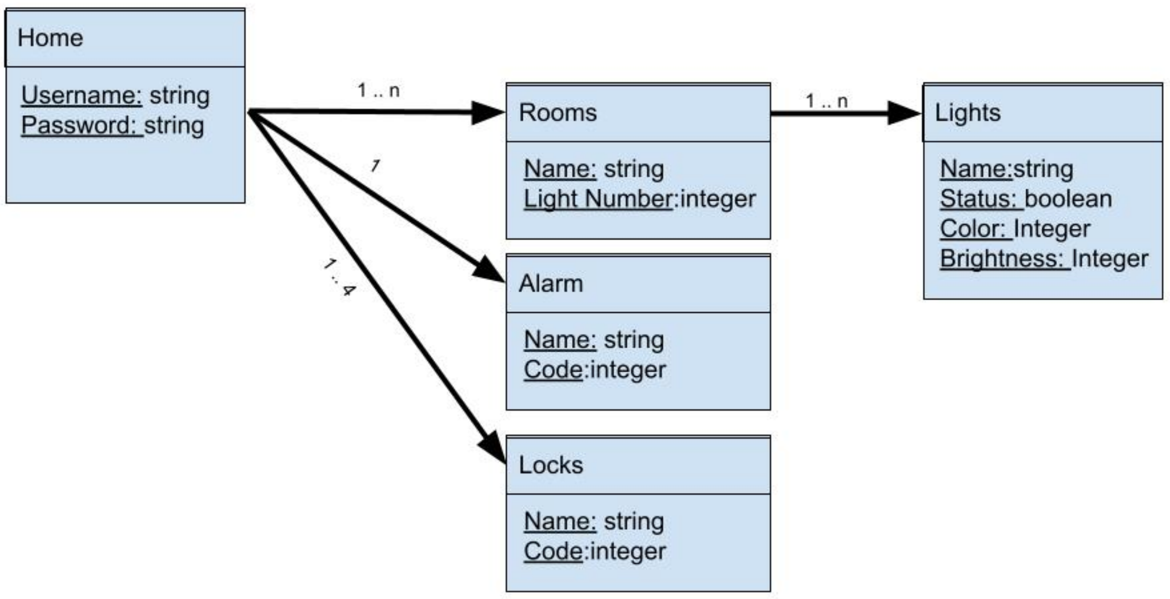


Figure 1: Data models follow the specifications

This is the simplest data model which we can use as a template given the fact that as we move one with this project we are going to be adding more features such as setting RGB colors,change lock combinations, setting the brightness, and later on be able to add more devices

4.1 Business Logic

4.1.1 Basic Operations

- Login/Logout
- Check Status of each device
- Change Status of each devices

4.1.2 Storyboard

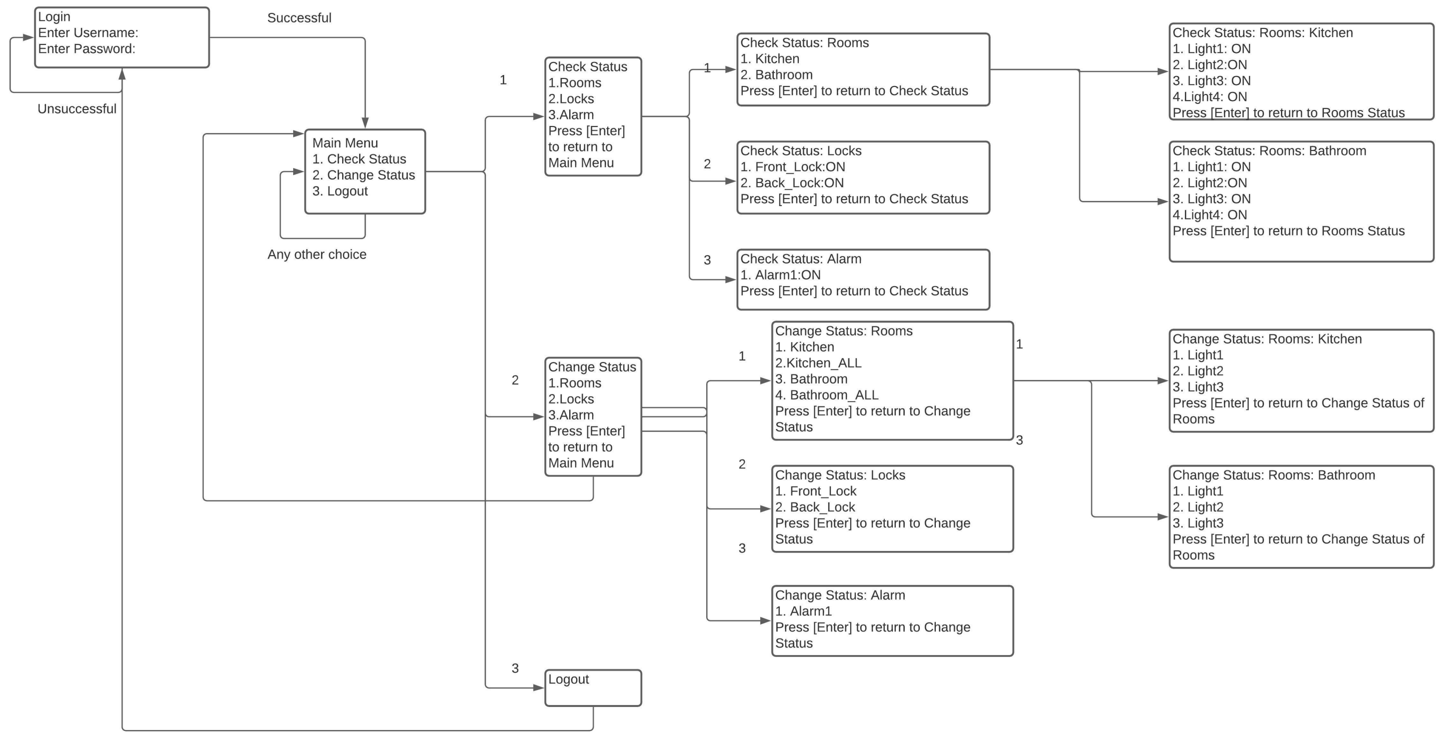
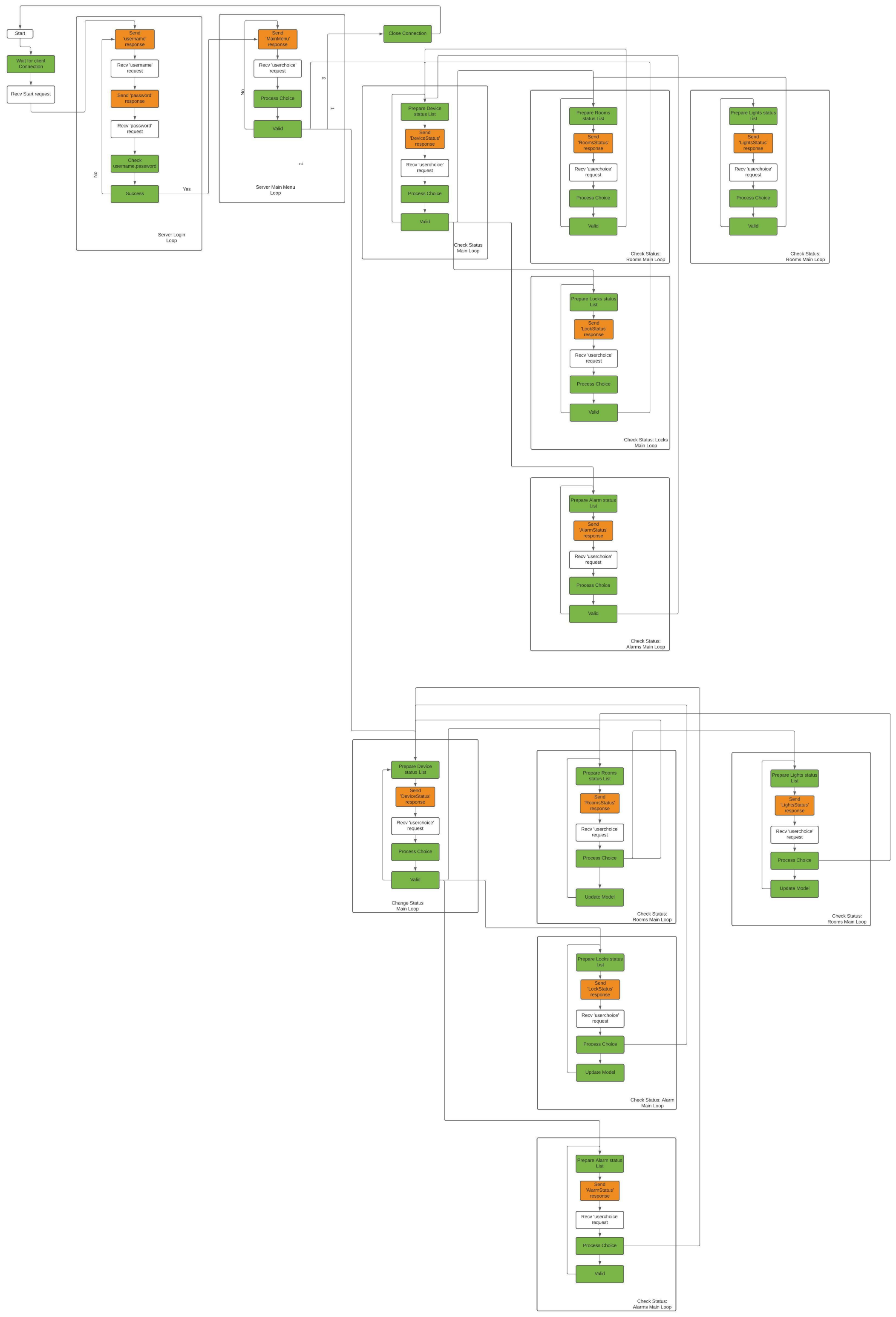


Figure 2:

The reason I chose to divide my check status and change status based on devices is because that is the method that my google home uses show the status of my devices and change the status of my devices. In my opinion the most effective method is by simply having a single main menu with a all the options since it leads to less amount of packets being transferred back and forth between the client and the server and can therefore reduce bandwidth. As we move on, I am going to implement more menus for the alarm,locks, and lights as way to provide more features for each device

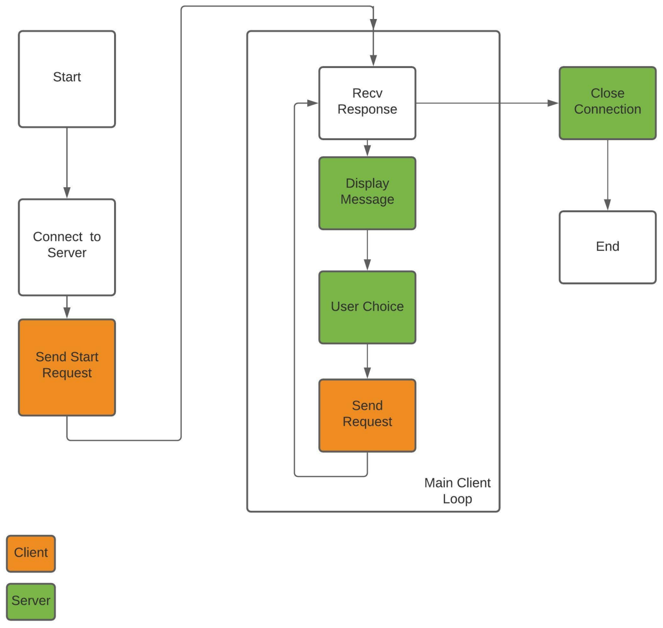
4.2 Sever state chart



Start
Wait for client Connection
Recv Start request
Send 'username' response
Recv 'username' request
Send 'password' response
Recv 'password' request
Check username,password
Success
No
Yes
Server Login Loop
Send 'MainMenu' response
Recv 'userchoice' request
Process Choice
Valid
No
1
2
3
Server Main Menu Loop
Close Connection
Prepare Device status List
Send 'DeviceStatus' response
Check Status Main Loop
Prepare Rooms status List
Send 'RoomsStatus' response
Check Status: Rooms Main Loop
Prepare Lights status List
Send 'LightsStatus' response
Prepare Locks status List
Send 'LockStatus' response
Check Status: Locks Main Loop
Prepare Alarm status List
Send 'AlarmStatus' response
Check Status: Alarms Main Loop
Change Status Main Loop
Update Model
Check Status: Alarm Main Loop

The server state diagram starts by simply waiting for the client to establish a connection. After establishing a connection, the user is going to be asked to send their username and password which is going to be check by the server based on the data model. If the username and password is correct then we move on to the main menu else we return back to the server login menu. After entering the main menu, the user needs to send the server a request for either checking status of a device, changing the status of a device, or logging out . After the user request the either check or change the status, they are going to receive the list of devices available to choose from. After requesting a device, the user is going to either the rooms, locks, or alarms available to them so they can either change or check the status. For changing the status there is an extra step in which the server updates the data model for each change and returns back to displaying the status of that device. For each menu there is always a return option which is simply by requesting to press enter.

4.3 Client State Chart



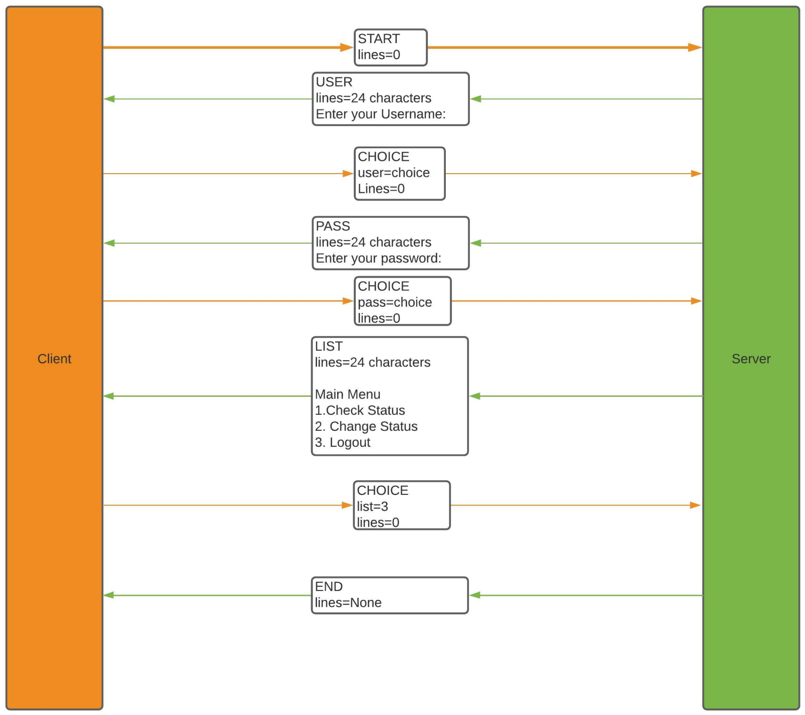
The client state diagram starts by simply establishing a connection between the server and the client. Then the server sends the client a message,which the user uses to make a request. After the request is sent to the server, the server makes a decision based on that request and executes a command. The connection between the server and the client ends when the user decides to log out.

4.4 Application Protocol Design

Client Application Protocols	Parameters	Body	Description
START	none	none	Establishes connection with the server
CHOICE	Lines=24 character,lines=0	None	Based on the options of the server, the user can send up to 24 character to the server
END	None	none	Ends the connection between the client and the server

Server Application Protocols	Parameters	Body	Description
USER	Lines=24 character	Line	Ask the user for the username
PASS	Line=24 character	Line	Ask the user for the password
LIST	Line=24 character	Line(s)	Gives the user options which they can chose from
ERROR	Line=24 character	Line(s)	Send the user information about an error in the system and waits for his choice
END	None	None	The server ends the connection between the client and the server

5 Example



6 Data model

```
#ifndef _DATAMODEL_H_
#define _DATAMODEL_H
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class light
{
public:
string name;
bool status;
};



class room
{
public:
vector<light> l;
};


```

```

class rooms
{
public:
vector<room> r;
string name;
int light_number;
};


class alarm
{
public:
string name;
int code;
};



class lock
{
public:
string name;
int code;
};



class locks
{
public:
vector<lock> llock;
};


class home
{
private:
string password;
string username;
public:
rooms n;
alarm a;
locks lo;
void setUsername(string newUsername);
void setPassword(string password);
};
#endif

```

7 Message Protocol

7.1 Client

7.1.1 Header file

```
#ifndef _CLIENT_MARSHAL_H_
#define _CLIENT_MARSHAL_H_
#include <iostream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include <vector>
using namespace std;
class server_message
{
public:
string options;
string message;
vector<string> lines;
int num_lines;
void printing()
{
cout<<options<<"    "<<num_lines<<"     "<<message<<endl;
for(unsigned int i=0;i<lines.size();i++)
{
cout<<lines[i]<<endl;
}

}
};

class client_message
{
public:
string options;
int decision;
void printing()
{
cout<<options<<"    "<<decision;
}
};

server_message unmarshal(string message)
{
string command;
server_message res;
stringstream ss(message);
ss>>command;
if(command=="USER")
```

```
{   res.options="USER";
getline(ss,res.message);
}
else if(command=="PASS")
{
res.options="PASS";
getline(ss,res.message);
}
else if(command=="LIST")
{
res.options="LIST";
getline(ss,res.message,'\\');
ss>>res.num_lines;
vector<stringstream> tt(res.num_lines);
string temp;
int i=0;
while(ss>>temp)
{
if(res.message=="\\\\")
{
break;
}
else if(temp=="\\")
{
i++;
}
else
{
tt[i]<<temp<<" ";
}
}
for(unsigned int j=0;j<tt.size();j++)
{
string temp2;
getline(tt[j],temp2);
res.lines.push_back(temp2);
}
}
else if(command=="ERROR")
{
res.options="ERROR";
getline(ss,res.message);
ss>>res.message;
}
else if(command=="END")
{
res.options="END";
getline(ss,res.message);
ss>>res.message;
}
else
```

```
{
}
return res;
}


string marshal(client_message cm)
{
stringstream ss;
string result;
if(cm.options=="START")
{
ss<<cm.options<<" "<<cm.decision;
}
else if(cm.options=="CHOICE")
{
ss<<cm.options<<" "<<cm.decision;
}
else if(cm.options=="END")
{
ss<<cm.options<<" "<<cm.decision;
}
else if(cm.options=="ERROR")
{
ss<<cm.options<<" "<<cm.decision;
}
else
{
}
getline(ss,result);
return result;
}
#endif

```

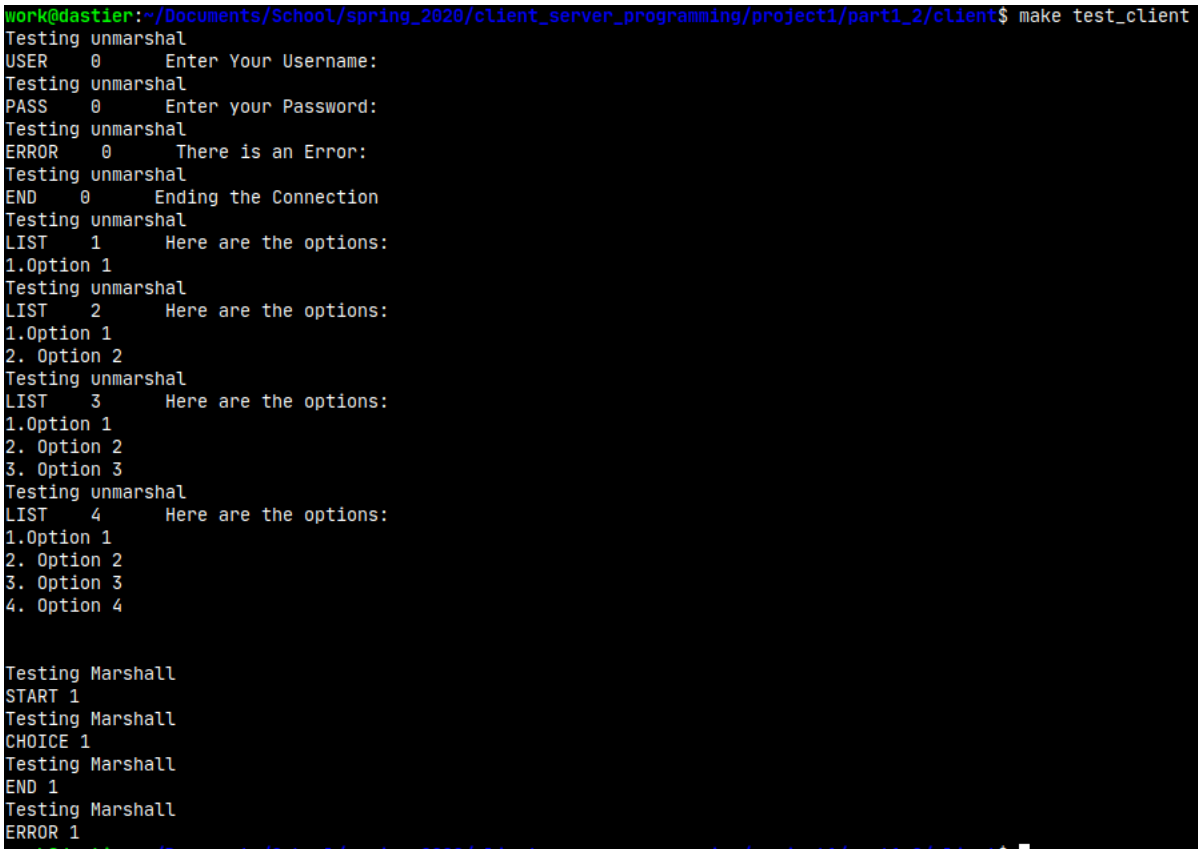
7.1.2 Test File

```
#include <iostream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include "client_marshal.h"
using namespace std;
int main(int argc,char*argv[])
{
if(argc==2)
{
cout<<"Testing unmarshal"<<endl;
server_message test1;
test1 =unmarshal(argv[1]);
test1.printing();
}

if(argc>2)
{
cout<<"Testing Marshall"<<endl;
client_message test;
test.decision=atoi(argv[2]);
test.options=argv[1];
cout<<marshal(test)<<endl;
}
return 0;
}

```

7.1.3 Makefile

```
all:client server_marshal.h client_marshal.h
echo Testing the client
client:client.cpp client_marshal.h
g++ client.cpp -o client
test_client: client client.cpp client_marshal.h
@./client "USER Enter Your Username:"
@./client "PASS Enter your Password:"
@./client "ERROR There is an Error:"
@./client "END Ending the Connection"
@./client "LIST Here are the options: \ 1 1.Option 1 \\"
@./client "LIST Here are the options: \ 2 1.Option 1 \ 2. Option 2 \\"
@./client "LIST Here are the options: \ 3 1.Option 1 \ 2. Option 2 \ 3. Option 3 \\"
@./client "LIST Here are the options: \ 4 1.Option 1 \ 2. Option 2 \ 3. Option 3 \ 4. Option 4 \\"
@ echo
@ echo
@./client "START" "1"
@./client "CHOICE" "1"
@./client "END" "1"
@./client "ERROR" "1"
clean:
rm client

```

7.2 Results



```
work@dastier:~/Documents/School/spring_2020/client_server_programming/project1/part1_2/client$ make test_client
Testing unmarshal
USER    0      Enter Your Username:
Testing unmarshal
PASS    0      Enter your Password:
Testing unmarshal
ERROR    0      There is an Error:
Testing unmarshal
END    0      Ending the Connection
Testing unmarshal
LIST    1      Here are the options:
1.Option 1
Testing unmarshal
LIST    2      Here are the options:
1.Option 1
2. Option 2
Testing unmarshal
LIST    3      Here are the options:
1.Option 1
2. Option 2
3. Option 3
Testing unmarshal
LIST    4      Here are the options:
1.Option 1
2. Option 2
3. Option 3
4. Option 4


Testing Marshall
START 1
Testing Marshall
CHOICE 1
Testing Marshall
END 1
Testing Marshall
ERROR 1
```

Figure 3: Results for the client marshalling

8 Server Marshal

8.1 Header File

8.2 Test program

8.3 Makefile