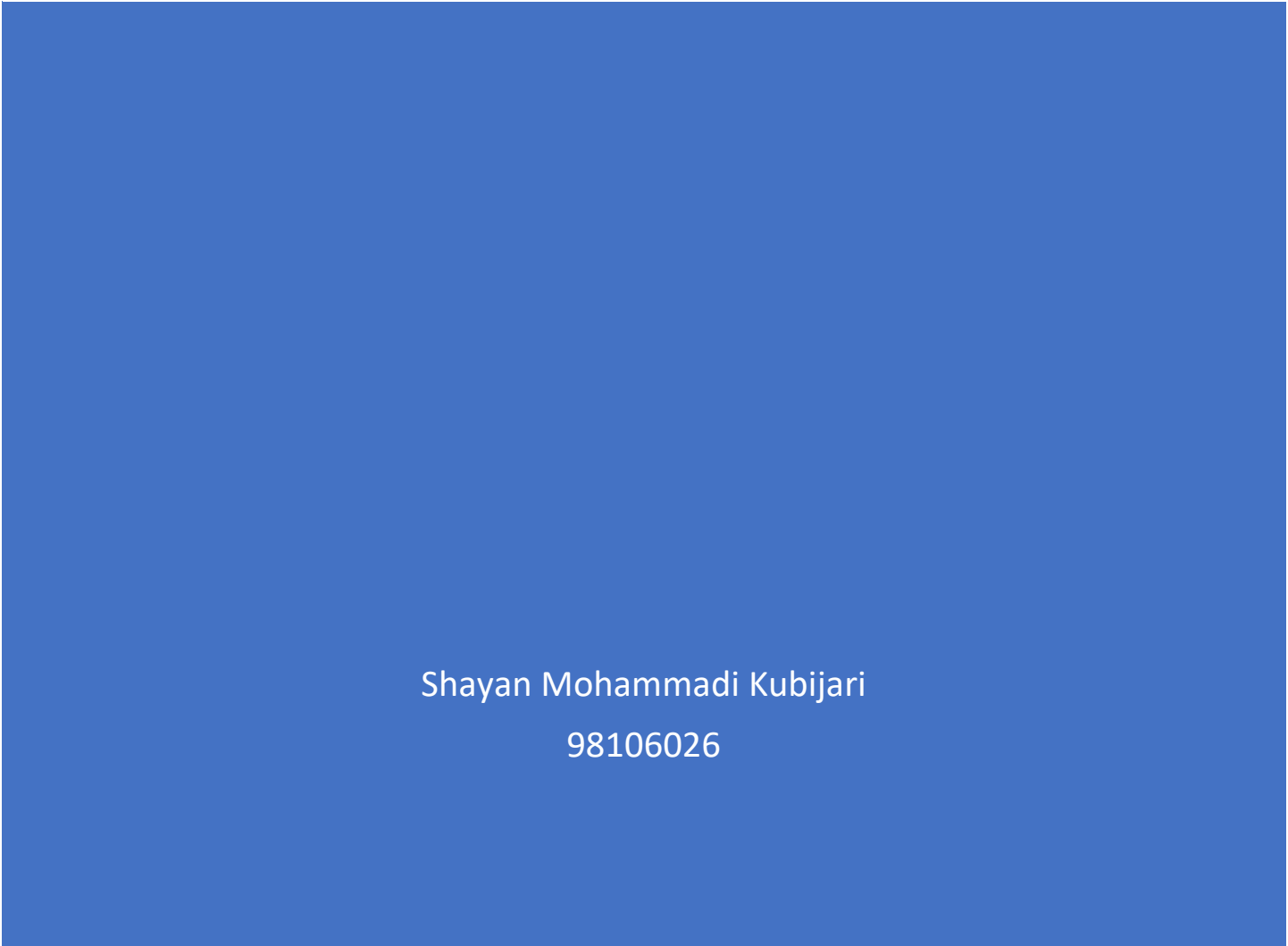




# CHAT APPLICATION

Fundamentals of Programming  
Fall 2019



Shayan Mohammadi Kubijari  
98106026

# Chat Application Project Document



Course: Fundamentals of Programming 98

Professor: Dr. Fakuri

By: Shayan Mohammadi Kubijari

---

Chat Application project includes two main parts; the Client and the Server, each containing several functions, which are connect and interact through "C Socket Programming" libraries, using json as a standard communication language.

This document provides a brief explanation for functions used in the client and server program, including inputs, output and general purpose of use.

## Table of Contents

1	Client .....	3
1.1	Global variables and Connection.....	3
1.2	Account Menu .....	4
1.3	Main Menu.....	4
1.4	Channel Menu .....	5
2	Server.....	6
2.1	Main.....	6
2.2	Global variables.....	7
2.3	Account.h .....	8
2.4	Channel.h .....	9
2.5	Database.h .....	10
3	my_cJSON.....	11
3.1	cJSON struct, the main focus .....	11
3.2	Public Functions.....	12
3.3	Private Functions.....	13

# 1 Client

Client application, basically, has 3 menus; “Account Menu”, “Main Menu” and “Channel Menu”.

- `void account_menu();`
- `void main_menu();`
- `void channel_menu();`

presenting menu options and redirecting to the appropriate function based on user's choice.

Subfunctions of these menus return a Boolean which represents the success of the function and controls the main while of these 3 menus.

\*\* In all the following functions, appropriate alert is provided in case of error response from server.

## 1.1 Global variables and Connection

- **AuthToken:** used for authentication in server and should be sent with every request after user signs in.
- **Buffer:** A single buffer is used for sending and receiving data between client and server, instead of defining a string every single time.
- `int socket_create();`

creates a socket and connects to server through default PORT (12345 in this case) and terminates the program if it is unable to connect.

This function outputs an integer which refers to the created socket and will be used in “winsock2.h” functions, like send and recv.

NOTE: in this program server and client will disconnect after every request.

## 1.2 Account Menu

- `void sign_up();`

Inputs username and password from user. a registration request will be sent to server and the response will be processed.

If there is no error, success message is shown to user.

User should sign in after registering.

- `bool sign_in();`

Inputs username and password and sends a login request to server and receives AuthToken, if not an error, which will be stored in a global string due to several usages in other functions.

After signing in user will be redirected to main menu.

- `exit(0)` terminates the program.

## 1.3 Main Menu

- `bool create_channel();`

Gets a channel name from user and sends a create channel request to server. AuthToken is used for client authentication in server and should be sent to server with any request after signing in.

If the process succeeds, user will be joined in the channel and redirected to channel menu automatically.

- `bool join_channel();`

Gets a channel name from user and sends a join channel request to server. Like create\_channel, user will go to channel menu if there is no error.

- `bool sign_out();`

a logout request will be sent to server (including the AuthToken).

If successful, user will sign out from its account and go back to account menu.

## 1.4 Channel Menu

Now that user is joined to a channel, it can send message, see other members names and messages or leave the channel.

- `void send_message();`

Gets a message from user and delivers a send message request to server (with AuthToken). Server store these messages in a database.

- `void refresh();`

Sends a refresh request to server and server sends back a list of sent messages in the channel as a response. Then `message_print()` function is called.

- `void message_print(cJSON* messages_list);`

A list of channel's messages is passed to this function by `refresh()`. This function reads and then prints list items one by one. (list size is counted by `cJSON_GetArraySize(messages_list)` ).

- `void members();`

Members request is sent to server and a member list is received as a response. Then, `members_print` function is called.

- `void members_print(cJSON* members_list);`

A list of channel's members is passed to this function by `members()`. This function reads and then prints list items one by one. (list size is counted by `cJSON_GetArraySize(members_list)` ).

- `bool leave_channel();`

Sends a leave request to server. If succeeded, user will automatically be redirected to main menu.

# 2 Server

## 2.1 Main

- `int socket_create();`

Creates a socket for accepting connections from server. Returns an integer which refers to server socket. This function works similar to its fellow in client program.

NOTE: client socket is also necessary for server to have (it is used in send and recv functions) because server is designed to connect and interact with several client through a single point.

The client socket is an output of “accept()” functions which is used after socket\_create() throughout the code.

- `void request_process();`

This function receives requests from client as a string. Then calls requested function by processing the string and stores a response cJSON from called functions' outputs. At the end, response is printed into a string and sent to the client.

**\*\* server's code(phase2) is divided into 3 C files other than main and their header files which is included in main(and in each other if needed). Header files contain functions' prototypes and general categorizing of the functions.**

There are some operations (mostly error check and prevention) which are common between upcoming functions, so they will be explained here to avoid repetition.

- Input checking: Buffer should be sent to these functions in a valid format. For example, for `sign_up()` it should be as follows:  
`<username>, <password>\n`  
if not, an “Invalid input pattern” error will be sent back to client and the function will not execute its main operations.
- AuthToken validation: there is an `AuthToken_check()` function in `channel.c` which is dedicated to checking the AuthToken with the valid ones stored in server. If not, “Invalid AuthToken” will be sent to client.

## 2.2 Global variables

- `users_path` and `channels_path`: These two strings define the location of the database which server will save users' and channels' data in it (e.g. `users_path/<username>.txt` ) . These will be initialized from `Server_Config.txt` when the program first starts.
- `cJSON *clients`: An cJSON object with following format:  
`{<AuthToken>: {“username”:<username>,”channel”:<ch_name>},...}`
- `cJSON *channels`: An cJSON object with following format:  
`{<ch_name>:[<member1>, <member2>,...],...}`



## 2.3 Account.h

- `char* AuthToken_generator(void);`

This function generates a 30 character random AuthToken with a-z, A-Z and 0-9 characters. It is only called by `sign_in()` function which gives an AuthToken to client, after a successful sign in, for later authentication.

- `cJSON* sign_up(const char* buffer);`

After error checking, if there is not one, function will check if username already exists (tries to open `<username>.txt` in database). If yes, "Username already exists" is sent back to client. Else, server will create a text file in database and stores the username and password in it (in json format).

- `cJSON* sign_in(const char* buffer);`

If `<username>.txt` doesn't exist, server returns "Username not found". Otherwise, an AuthToken will be generated and sent to client.

Server also, keeps the AuthToken in an cJSON Object, named "clients", to keep track of online users.

```
cJSON* client = cJSON_AddObjectToObject(clients, AuthToken);  
cJSON_AddStringToObject(client, "username", username);
```

- `cJSON* sign_out(const char* buffer);`

After AuthToken validation check, appropriate response is sent to client and AuthToken will be removed from clients object.

```
cJSON_DeleteItemFromObject(clients, AuthToken);
```

## 2.4 Channel.h

- `void AuthToken_check(cJSON* response, const char* AuthToken);`

As said earlier, this functions check if AuthToken is valid or no. it adds appropriate error to response if so.

- `void channel_check(cJSON* response, const char* AuthToken);`

This function is for checking if user is a member of any channel or no. if not, an error is added to response. (It is not called in create and join)

- `cJSON* create_channel(const char* buffer);`

First, checks if channel already exists or not (error will be sent back if so).

Then creates a new file in database with channel's name.

User will automatically join the channel.

Channel also added to channels object and client's data.

- `cJSON* join_channel(const char* buffer);`

Checks if channel exists. Then adds channel to client's data and channels object. Also, successful message is sent to client if the process succeeds.

- `cJSON* send_message(const char* buffer);`

Gets message and its sender from buffer. Then opens the channel's data file (in a cJSON array of objects) and appends message to it: (following format)

Write\_data() is called to write data in channel's file.

```
[{"sender1":"message1"}, {"sender2":"message2"}, ...]
```

- `cJSON* refresh(const char* buffer);`

reads messages from channel's file and sends messages array as a response to client( if all errors are passed successfully).

- `cJSON* members(const char* buffer);`

sends members list array for the client:

`cJSON_GetItemFromObject(channels, ch_name);`

- `cJSON* leave_channel(const char* buffer);`

Removes client from channel's member list and removes channel from clients data.

## 2.5 Database.h

These are functions for managing database.

- `void make_dir(char* path);`

Makes directory recursively. E.g. Resources/users : makes Resources and the makes users inside it.

- `void init_database(const char* config_path);`

This function initializes the server from Server\_Config.txt when the program starts.

- `cJSON* read_data(const char* path);`

- `void write_data(cJSON*, const char* path);`

These two functions read and write json data from and to a file.

NOTE: `read_data` allocates memory to create the cJSON, so you should make sure to free the allocated memory and avoid memory leakage.

# 3 my\_cJSON

As a part of phase 3, we were supposed to rewrite cJSON functions, which were used in our codes, by ourselves.

## 3.1 cJSON struct, the main focus

cJSON uses a brilliant tree shaped struct with following definition: (it is a little bit modified due to personal use)

```
typedef struct cJSON{  
    // next and previous items in an object/array  
    struct cJSON* next;  
    struct cJSON* prev;  
    // child of an object/array (initializes with NULL)  
    struct cJSON* child;  
    // item's type; object(0), array(1) or string(2)  
    int type;  
    // cJSON_Print() output size  
    int print_size;  
    // only string values are supported  
    char* valuelstring;  
    // object's name (if it is child or member of another object/array)  
    char* name;  
} cJSON;
```

Every subgroup of items starts with a child(parent->child) and continues through next and prev pointers. e.g.:

```
{"item1":"value1","item2":"value2",...}
```

Item1 is the child of parent object and item to in item1's next (item1 is item2's prev).

NOTE: in my\_cJSON 3 item types are defined: Object(0), Array(1) and string(2).

## 3.2 Public Functions

To prevent the necessity of changing all the cJSON functions used in server and client codes, my\_cJSON is completely compatible with cJSON and share the same function names with it and the serve the exact same functionality. A list of supported functions comes below:

// Create functions

- cJSON\* cJSON\_CreateObject(void);
- cJSON\* cJSON\_CreateArray(void);
- cJSON\* cJSON\_CreateString(const char\* string);

// Add functions

- cJSON\* cJSON\_AddItemToObject(cJSON\* object, char\* name, cJSON\* item);
- cJSON\* cJSON\_AddItemToArray(cJSON\* array, cJSON\* item);
- cJSON\* cJSON\_AddArrayToObject(cJSON\* object, const char\* name);
- cJSON\* cJSON\_AddObjectToObject(cJSON\* object, const char\* name);
- cJSON\* cJSON\_AddStringToObject(cJSON\* object, char\* name, char\* string);

// Get functions

- cJSON\* cJSON\_GetArrayItem(cJSON\* array, int index);
- int cJSON\_GetArraySize(cJSON\* array);
- cJSON\* cJSON\_GetObjectItem(cJSON\* object, const char\* name);

- `char* cJSON_GetStringValue(cJSON* item);`  
// Delete functions
- `void cJSON_Delete(cJSON* item);`
- `void cJSON_DeleteItemFromArray(cJSON* array, int index);`
- `void cJSON_DeleteItemFromObject(cJSON* object, const char* name);`  
// Print, Parse and Duplicate
- `char* cJSON_PrintUnformatted(cJSON* item);`
- `cJSON* cJSON_Parse(char* string);`

### 3.3 Private Functions

There are some functions which are used internally in my\_cJSON.c and have place in public usage of the library.

- `cJSON* create_item(int create_type);`

For creating all 3 types of items.

- `char* print(cJSON* item, bool main_output);`
- `void print_object(cJSON* object);` // printing an object
- `void print_array(cJSON* array);` // printing an array
- `void print_string(cJSON* string);` //printing a string
- `cJSON* parse(void);`
- `cJSON* parse_object(void);` // parsing an object
- `cJSON* parse_array(void);` // parsing an array
- `cJSON* parse_string(void);` // parsing a string