In the Name Of God

CGram Documentation

Contains CGram and CGram Server as a part of Phase 3

Designed and developed by Seyed Parsa Neshaei - 98106134

Fundamentals of Programming, Semester #1, 98-99

Sharif University of Technology
Department of Computer Engineering

CGram Client Documentation

Most important methods and properties

Controller Variables, Definitions, Constants, Structures and Enumerations

These properties, structures and enumerations are used to control how the app is displayed and worked with in the Terminal.

1. `#define MAX 200000`

   This definition sets a maximum value for array size limit in many parts of the code.

2. `#define PORT 12345`

   This is the standard port used throughout the project in all phases for connecting the server and the client.

3. `#define SA struct sockaddr`

   This is a redefinition of "struct sockaddr" using a new name, "SA".

4. `char server_socket;`

   This is the global server socket identifier used by Network Functions to connect to the server.

5. `char asciiArt[5][200];`

   This is an array of strings, used in presenting ascii art to the users when they start using CGram.

6. `char username[100];`

   A string indicating the current user's username.

   NOTE CGram doesn't save the user's password like the user's username and in the same manner, due to security goals set when starting to develop the app.

7. `char token[400];`

   A string indicating the current user's token, provided by the server when logged in.

8. `char channel[100];`

   A string indicating the current joined channel's name.

9. `typedef struct { char name[200]; } Member;`

   A structure representing members in a channel.

Parameters and Members

— name: The name of the member.

10. `typedef struct { char sender[200]; char content[200]; } Message;`

A structure representing messages in a channel.

Parameters and Members

— sender: The name of the member who has sent the message.

— content: The content of the message.

11. `Member members[200];`

An array containing the members of the current channel which the client is joined into.

12. `int membersCount = 0;`

An integer holding the current count of the members array, used due to dynamic nature of the array.

13. `Message messages[200];`

An array containing the messages of the current channel which the client is joined into.

14. `int messagesCount = 0;`

An integer holding the current count of the messages array, used due to dynamic nature of the array.

15. `static struct termios term, oterm;`

Two TERM-related variables which are needed for getch() function from the Controller Functions to work correctly.

16. `int terminalColumns = 80, terminalLines = 24;`

Two variables which hold the dimensions of the Terminal and allow printStringCentered() from the Controller Functions to work correctly.

17. `enum Colors { RED = 35, GREEN = 32, YELLOW = 33, BLUE = 34, CYAN = 36 } appColor = RED;`

The enumeration which defines 5 colors to be used in app's main theme.

18. `enum PasswordStatus { TOOWEAK = 1, WEAK = 2, MODERATE = 3, STRONG = 4, TOOSTRONG = 5, TOOSHORT = 6 };`

The enumeration which defines 6 different types of password status to be used in app's register page.

19. `char requestToSend[MAX];`

The string containing the request which should be sent to the server, preserved as a global variable.

```
20. char originalReq[MAX];
```

The string containing the original value of request. This global variable is used in copyReqs().

```
21. int client_socket;
```

The client socket's identifier, preserved as a global variable.

Controller Functions

These functions are used to control how the app is displayed and worked with in the Terminal.

```
1. static int getch(void);
```

Scans the next pressed character on the keyboard by the user.

Parameters

None

Return Value

An integer representing the ASCII code of the character mapped to the pressed key on the keyboard.

```
2. void setupTerminalDimensions(void);
```

Sets the global variables "terminalColumns" and "terminalLines" to represent correct data.

Parameters

None

Return Value

None

```
3. void printStringCentered(char *str);
```

Prints the given string centered in the Terminal.

The function gets use of the global variables "terminalColumns" and "terminalLines" to find out how it should invoke printf() to print the given string centered.

Parameters

— *str: The string to be printed centered.

Return Value

None

```
4. void hideCursor(void);
```

Hides the flashing cursor in the Terminal.

The function gets use of special Terminal codes and prints them to achieve this functionality.

Parameters

None

Return Value

None

5. `void showCursor(void);`

Shows the flashing cursor in the Terminal.

The function gets use of special Terminal codes and prints them to achieve this functionality.

Parameters

None

Return Value

None

6. `void makeBoldColor(void);`

Makes the text being typed bold and colored.

The function gets use of special Terminal codes and prints them to achieve this functionality.

Parameters

None

Return Value

None

7. `void makeColor(void);`

Makes the text being typed colored.

The function gets use of special Terminal codes and prints them to achieve this functionality.

Parameters

None

Return Value

None

8. `void resetFont(void);`

Makes the text being use the default font.

The function gets use of special Terminal codes and prints them to achieve this functionality.

Parameters

None

Return Value

None

```
9. int main(int argc, const char * argv[]);
```

Marks the starting point of the app.

The function first setups the necessary global variables by calling the respective functions and then checks the parameters passed to it, namely argc and argv, to find out which command line arguments are passed, to call the necessary functions to continue the procedure. If no important argument is passed, the function calls welcomePage() to take care of showing the starter page of the app.

Parameters

— argc: The number of passed command line arguments.

— * argv[]: An array of strings, containing passed command line arguments.

Return Value

An integer indicating the app has been run successfully (0) or not (any other number) to be used by the caller / operating system.

Data Manipulation Functions

These functions process the data in the app (especially arrays and strings) to produce the right output.

```
1. void splitString(char str[1000000], char newString[1000]
[1000], int *countOfWords);
```

Splits the string given into parts separated by space.

The function traverses the string, looking for spaces, and separates the whole string into smaller parts to be accessible for the user more easily.

Parameters

— str[]: The string to be splitted.

— newString[1000][]: The array of strings used to pour processed data into.

— *countOfWords: A pointer to an integer containing the count of words after the string being split. Used in looping over newString[1000][].

Return Value

None

```
2. void splitStringByDoubleQuotes(char str[1000000], char
newString[1000][1000], int *countOfWords);
```

Splits the string given into parts separated by “”.

The function traverses the string, looking for “”, and separates the whole string into smaller parts to be accessible for the user more easily.

Parameters

— str[]: The string to be splitted.

— newString[1000][]: The array of strings used to pour processed data into.

— *countOfWords: A pointer to an integer containing the count of words after the string being split. Used in looping over newString[1000][].

Return Value

None

```
3. void findSubstring(char *destination, const char *source,
int beg, int n);
```

Extracts a specified count characters from the source string, starting from the beginning index provided.

The function starts from the beginning index and copies characters one-by-one into the destination string.

Parameters

— *destination: The string which should contain the result of extraction.

— *source: The string which substring should be extracted from.

Return Value

None

```
4. char *replaceWord(const char *s, const char *oldW, const
char *newW);
```

Replaces all occurrences of a substring with a new one throughout the given source string.

The function uses the <string.h>-provided strstr() function to achieve this functionality.

Parameters

— *s: The source string.

— *oldW: A string which tells the function which word it should look for to replace.

— *newW: A string containing the new word which the old word should be replaced by.

Return Value

A string containing the result of the replacement procedure.

5. `int getWords(char *base, char target[10][200]);`

Separates the words of the given sentence.

The function loops over the string's characters and splits the sentence using the space separator.

Parameters

— *base: The source string.

— target[10][]: An array of strings, indicating different words after the source string being split

Return Value

An integer indicating the number of words split.

6. `int strStartsWith(const char *pre, const char *str);`

Checks that the given string includes the given prefix or not.

The function uses memcmp() to compare strings and produce the correct result.

Parameters

— *pre: The prefix string.

— *str: The source string.

Return Value

An integer indicating if the given source starts with the given prefix; the function returns 1 if yes, and 0 if no.

7. `enum PasswordStatus passwordStatus(char *password);`

Determines the given password's strength.

The function checks the length, the number of digits and special characters and the number of uppercase letters of the given password to determine its strength.

Parameters

— *password: The given password to be checked.

Return Value

The strength status of the given password.

8. `void copyReqs(void);`

Copies originalReq[] into requestToSend[].

Parameters

None

Return Value

None

9. `void initializeAsciiArt(void);`

Fills the global asciiArt[] array of strings by the ASCII art pre-made for the word 'CGRAM'.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

None

Return Value

None

10. `void initializeAppColor(void);`

Sets the global appColor enumeration variable to its correct data provided by the 'appcolor.txt' file, which contains the user's preference.

The global appColor set in this function is respected by makeBoldColor() and makeColor() functions by making text bold and colored or only colored, respectively.

If 'appColor.txt' is non-existent, the function defaults to the red color. This is the option mainly provided by the app's developer.

Parameters

None

Return Value

None

Network Functions

These functions have the role of connecting to the server, sending requests and receiving responses on demand.

1. `void chat(int server_socket);`

Sends and receives data using the given socket identifier.

After zero-ing the request variable, this function attempts to receive new data from the server.

Parameters

— server_socket: The server socket's identifier.

Return Value

None

2. `void sendRequestToServer(char *request, char *result);`

Connects to the server, and calls chat() to start exchanging data between the server and the client.

The function uses sockaddr_in to build a socket and then it connects to the given port and IP address defined previously. After calling chat(), it attempts to shut the socket down.

Parameters

— server_socket: The server socket's identifier.

Return Value

None

Parsing Functions

These functions parse and process the server responses received from the Network Functions and turn them into a recognizable format for UI functions.

1. `int loginServer(const char username[], const char password[]);`

Builds the request needed to login, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status and the user's token from it.

There is a second version of this function with the signature loginServer_cJSON(const char username[], const char password[]), which uses the cJSON library to do the exact same tasks.

Parameters

— username[]: The user's username.

— password[]: The user's password.

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: unknown error

— -2: wrong password

— -3: not-existent username

2. `int registerServer(char username[], char password[]);`

Builds the request needed to register, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status from it.

There is a second version of this function with the signature registerServer_cJSON(char username[], char password[]), which uses the cJSON library to do the exact same tasks.

Parameters

— username[]: The user's username.

— password[]: The user's password.

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: used username

3. `int createChannelServer(void);`

Builds the request needed to create a channel, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status from it.

There is a second version of this function with the signature createChannelServer_cJSON(void), which uses the cJSON library to do the exact same tasks.

Parameters

None

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: used channel name

4. `int findChannelServer(void);`

Builds the request needed to join a channel, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status from it.

There is a second version of this function with the signature findChannelServer_cJSON(void), which uses the cJSON library to do the exact same tasks.

Parameters

None

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: non-existent channel name

```
5. int logoutServer(void);
```

Builds the request needed to log out, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status from it.

There is a second version of this function with the signature logoutServer_cJSON(void), which uses the cJSON library to do the exact same tasks.

Parameters

None

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: authentication failed

```
6. int reloadMembersServer(void);
```

Builds the request needed to fetch current channel's members, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status and the name of the members from it.

There is a second version of this function with the signature reloadMembersServer_cJSON(void), which uses the cJSON library to do the exact same tasks.

Parameters

None

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: unknown error

```
7. int searchMessageServer(char *word);
```

Builds the request needed to search between current channel's messages, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status and the details of the found messages from it.

Parameters

— *word: The word for which the matching messages should be presented in the app.

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: unknown error

```
8. int reloadMessagesServer(void);
```

Builds the request needed to fetch current channel's messages, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status and the details of the messages from it.

There is a second version of this function with the signature reloadMessagesServer_cJSON(void), which uses the cJSON library to do the exact same tasks.

Parameters

None

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: unknown error

```
9. int sendMessageServer(char message[]);
```

Builds the request needed to add a new message to the current channel's messages, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status from it.

There is a second version of this function with the signature sendMessageServer_cJSON(void), which uses the cJSON library to do the exact same tasks.

Parameters

— message[]: The content of message to be added to the current channel's messages. The sender of the message is determined automatically.

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: unknown error

```
10. int searchMemberServer(char *memberName);
```

Builds the request needed to search for a member of the current channel, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status from it.

Parameters

— *memberName: The name of the member to be searched for in the current channel's members.

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: non-existent member

```
11. int leaveChannelServer();
```

Builds the request needed to leave the current channel, calls network functions needed for completing the necessary tasks and then parses the incoming JSON to extract success status from it.

There is a second version of this function with the signature leaveChannelServer_cJSON(void), which uses the cJSON library to do the exact same tasks.

Parameters

None

Return Value

An integer indicating the success of the action done. Possible values for this integer are as follows:

— 0: success

— -1: not joined in any channel to leave / unknown error

UI Functions

These functions present the app's UI, utilizing different menus and options for users to select from to achieve their goal in using the app more quickly.

```
1. void loginPage(void);
```

Shows the login page of the app.

The function gets the user's username and password, and then tries to login using loginServer().

NOTE Because CGram inputs data centered in the screen, it uses a helper function, loginPage_printUntilSpecifiedUsername(char *username), which gets the username typed up to the point the function is called, and presents a new screen with the username printed in. Username is grabbed character-by-character and after each letter has been typed, the latter function is called.

Parameters

None

Return Value

None

2. `void registerPage(void);`

Shows the register page of the app.

The function gets the user's username and password, and then tries to register using registerServer(). Also, this function gets use of passwordStatus() to show the password strength live in the register page.

NOTE Because CGram inputs data centered in the screen, it uses a helper function, registerPage_printUntilSpecifiedUsername(char *username), which gets the username typed up to the point the function is called, and presents a new screen with the username printed in. Username is grabbed character-by-character and after each letter has been typed, the latter function is called. Also, because the password should also be typed centered, the helper function registerPage_printUntilSpecifiedPassword(char *username, char *password) is used, too.

Parameters

None

Return Value

None

3. `void mainPage(void);`

Shows the main page of the app after login.

The function allows the user to join or create channels and they could also easily exit the main page using the "logout" command, which functionality is provided in this function.

NOTE Because CGram inputs data centered in the screen, it uses a helper function, mainPage_printUntilSpecifiedChannel(char *channel), which gets the commands typed up to the point the function is called, and presents a new screen with the command printed in. Command

is grabbed character-by-character and after each letter has been typed, the latter function is called.

Parameters

None

Return Value

None

4. `void chatPage(void);`

Shows the chat page of the app after joining a channel.

The function allows the user to send a new message, find messages including a certain word, reload messages quickly, see all members of the channel and search between their names and also leave the channel on demand.

The function also presents all chats done in the channel up to now in a thoughtful style.

Parameters

None

Return Value

None

5. `void colorChangePage(void);`

Shows the color picker page of the app.

The function allows the user to change the main app's theme color on demand, just by pressing 'c' in the welcome page. The user's preference is saved into a file named 'appcolor.txt' and is loaded every time the app is launched.

The function gets use of enum Colors to set the app's main theme color.

Parameters

None

Return Value

None

6. `void welcomePage(void);`

Shows the starter / welcome page of the app.

The function introduces the app to new user and allows them to login, register and change the app's theme color by just pressing a key on the keyboard.

The function also features quit confirmation to prevent cases of unexpected quit.

The function does its job by correctly identifying the user's needs and calling other functions whenever necessary.

The initialized ASCII art is also printed in this function.

Parameters

None

Return Value

None

```
7. void commandLineLogin(const char *username, const char
*password);
```

Shows the login page of the app when login is invoked by the command line arguments.

The function attempts to login quickly, and in the case of error, it provides the option to return back to the Terminal or to continue using the app by going to the welcome page.

The function also features quit confirmation to prevent cases of unexpected quit.

The login procedure is done by calling other functions whenever necessary- for example, loginServer() - .

Parameters

None

Return Value

None

```
8. void commandLineInvalid(void);
```

Shows an error page when command line arguments are invalid.

The function is invoked by main(), whenever the number or the contents of the command line arguments are unexpected, and shows an error message detailing how to get help for using CGram command line arguments.

Parameters

None

Return Value

None

```
9. void commandLineHelp(void);
```

Shows the manual page of the app's command line interface (CLI).

The function is invoked whenever '—help' are passed and provides the necessary knowledge base needed for the users of the app.

Parameters

None

Return Value

None

CGram Server Documentation

Most important methods and properties

Controller Variables, Definitions, Constants, Structures and Enumerations

These properties, structures and enumerations are used to control how the app is displayed and worked with in the Terminal.

1. `#define MAX 1000000`

    This definition sets a maximum value for array size limit in many parts of the code.

2. `#define PORT 12345`

    This is the standard port used throughout the project in all phases for connecting the server and the client.

3. `#define SA struct sockaddr`

    This is a redefinition of "struct sockaddr" using a new name, "SA".

4. `typedef struct User { char username[1000]; char token[20]; char currentChannel[1000]; } User;`

    A structure representing users online and logged in.

    Parameters and Members

    — username: The user's username.

    — token: The generated token by generateToken() to accompany the user. The token is checked against every time a logged in client sends a request.

    — currentChannel: The name of the channel which the user is joined into. If the user hasn't joined any channel, strcmp(currentChannel, "") evaluates to zero.

5. `static struct termios term, oterm;`

    Two TERM-related variables which are needed for getch() function from the Controller Functions to work correctly.

6. `User allUsers[MAX];`

    An array containing all the users online and logged in.

7. `int currentAllUsersIndex = 0;`

An integer containing the number of users online and logged in. This is used when looping over allUsers[].

8. `char asciiArt[5][200];`

This is an array of strings, used in presenting ascii art to the users when they start using CGram.

9. `enum Colors { RED = 35, GREEN = 32, YELLOW = 33, BLUE = 34, CYAN = 36 } appColor = RED;`

The enumeration which defines 5 colors to be used in app's main theme.

10. `int terminalColumns = 80, terminalLines = 24;`

Two variables which hold the dimensions of the Terminal and allow printStringCentered() from the Controller Functions to work correctly.

Controller Functions

These functions are used to control how the app is displayed and worked with in the Terminal.

1. `static int getch(void);`

Scans the next pressed character on the keyboard by the user.

Parameters

None

Return Value

An integer representing the ASCII code of the character mapped to the pressed key on the keyboard.

2. `void setupTerminalDimensions(void);`

Sets the global variables "terminalColumns" and "terminalLines" to represent correct data.

Parameters

None

Return Value

None

3. `void writeToFile(char *fileName, char *data);`

Writes the given string into the given text file address.

The function gets use of FILE structure to achieve its goal.

Parameters

— *fileName: The string containing text file's address.

— *data: The string which should be written to the text file.

Return Value

None

4. `void readFromFile(char *fileName, char *data);`

Reads the contents of the given text file address.

The function gets use of FILE structure to achieve its goal.

Parameters

— *fileName: The string containing text file's address.

— *data: The string which will include text file's data after it has been read.

Return Value

None

5. `void makeBoldColor(void);`

Makes the text being typed bold and colored.

The function gets use of special Terminal codes and prints them to achieve this functionality.

Parameters

None

Return Value

None

6. `void makeColor(void);`

Makes the text being typed colored.

The function gets use of special Terminal codes and prints them to achieve this functionality.

Parameters

None

Return Value

None

7. `void resetFont(void);`

Makes the text being use the default font.

The function gets use of special Terminal codes and prints them to achieve this functionality.

Parameters

None

Return Value

None

8. `void printStringCentered(char *str);`

Prints the given string centered in the Terminal.

The function gets use of the global variables “terminalColumns” and “terminalLines” to find out how it should invoke printf() to print the given string centered.

Parameters

— *str: The string to be printed centered.

Return Value

None

9. `int main(int argc, const char * argv[]);`

Marks the starting point of the app.

The function first setups the necessary global variables by calling the respective functions and then checks the parameters passed to it, namely argc and argv, to find out which command line arguments are passed, to call the necessary functions to continue the procedure. If no important argument is passed, the function calls connectServer() to take care of preparing the server to accept requests from the clients.

Parameters

— argc: The number of passed command line arguments.

— * argv[]: An array of strings, containing passed command line arguments.

Return Value

An integer indicating the app has been run successfully (0) or not (any other number) to be used by the caller / operating system.

Data Manipulation Functions

These functions process the data in the app (especially arrays and strings) to produce the right output.

1. `char *replaceWord(const char *s, const char *oldW, const char *newW);`

Replaces all occurrences of a substring with a new one throughout the given source string.

The function uses the <string.h>-provided strstr() function to achieve this functionality.

Parameters

— *s: The source string.

— *oldW: A string which tells the function which word it should look for to replace.

— *newW: A string containing the new word which the old word should be replaced by.

Return Value

A string containing the result of the replacement procedure.

```
2. void insertString(char *destination, int pos, char
*stringToInsert);
```

Inserts the given string into the given position of the given source string.

The function uses the <string.h>-provided strncpy(), strcat() and strcpy() functions to achieve this functionality, using dynamic memory allocation.

Parameters

— *destination: The source string. This array of characters is manipulated inside the function.

— pos: An integer indicating the position to insert the substring into.

— *stringToInsert: The string that should be inserted in the position 'pos'.

Return Value

None

```
3. void occurences(char *str, char *toSearch, int
*numberOfOccurences, int occurencesArr[500]);
```

Finds all the occurrences of the given term in the given string.

The function uses the <string.h>-provided strlen() function to achieve this functionality.

This function is mostly needed when parsing JSON without using any library.

Parameters

— *str: The source string.

— *toSearch: The term to search for inside the source string.

— *numberOfOccurences: A pointer to an integer indicating the number of times the term was found in the source string.

— occurencesArr[]: An array of integers indicating the positions in which the words are found.

Return Value

None

4. `void splitStringWithoutSpace(char str[1000000], char newString[1000][1000], int *countOfWords);`

Splits the string given into parts separated by comma, without taking spaces in account.

The function traverses the string, looking for commas, and separates the whole string into smaller parts to be accessible for the user more easily.

Parameters

— str[]: The string to be splitted.

— newString[1000][]: The array of strings used to pour processed data into.

— *countOfWords: A pointer to an integer containing the count of words after the string being split. Used in looping over newString[1000][].

Return Value

None

5. `void splitStringByDoubleQuotes(char str[1000000], char newString[1000][1000], int *countOfWords);`

Splits the string given into parts separated by “”.

The function traverses the string, looking for “”, and separates the whole string into smaller parts to be accessible for the user more easily.

Parameters

— str[]: The string to be splitted.

— newString[1000][]: The array of strings used to pour processed data into.

— *countOfWords: A pointer to an integer containing the count of words after the string being split. Used in looping over newString[1000][].

Return Value

None

6. `void splitString(char str[1000000], char newString[1000][1000], int *countOfWords);`

Splits the string given into parts separated by space and comma.

The function traverses the string, looking for spaces and commas, and separates the whole string into smaller parts to be accessible for the user more easily.

Parameters

— str[]: The string to be splitted.

— newString[1000][]: The array of strings used to pour processed data into.

— *countOfWords: A pointer to an integer containing the count of words after the string being split. Used in looping over newString[1000][].

Return Value

None

7. `void generateToken(char *token, int size);`

Generates a random token of the given size.

The function utilized the rand() function to pick random characters from an inside-function constant charset[]. The function also takes care of null-terminating the token.

Parameters

— *token: The string which will include the token after it has been generated.

— size: The size of the token to be generated.

Return Value

None

8. `void findSubstring(char *destination, const char *source, int beg, int n);`

Extracts a specified count characters from the source string, starting from the beginning index provided.

The function starts from the beginning index and copies characters one-by-one into the destination string.

Parameters

— *destination: The string which should contain the result of extraction.

— *source: The string which substring should be extracted from.

Return Value

None

9. `int stringContainsWord(char *string, char *word);`

Determines whether the given string includes the given word.

The function utilizes the already-defined splitString() function to get a list of words of the string, and then starts to loop over the resulted list to find the given word.

Parameters

— *string: The source string.

— *word: The word to be looked for in the string.

Return Value

An integer, which is 1 when the string contains the word, and 0 otherwise.

10. `int numberOfOccurencesOfWordInString(char *word, char *string);`

Counts the number of occurrences of the given word within the given string.

The function utilizes the already-defined splitString() function to get a list of words of the string, and then starts to loop over the resulted list to count how many times the word is repeated.

Parameters

— *word: The word to be looked for in the string.

— *string: The source string.

Return Value

An integer indicating the number of occurrences of the given word within the given string.

11. `int userHasGivenPasswordInJSON(char *user, char *pass, char *root);`

Determines if the given user's password is correct or not.

The function looks into the provided JSON string and searches for special JSONy signs, like double quotes, braces or brackets.

There is a second version of this function with the signature userHasGivenPasswordInJSON_cJSON(char *user, char *pass, cJSON *root), which uses the cJSON library to do the exact same tasks.

Parameters

— *user: The user's username.

— *pass: The user's password.

— *root: The JSON string the function should look into. The string represents the contents / parts of the contents of 'users.txt'.

Return Value

An integer indicating if the password is correct (1) or not (0).

12. `int memberExistsInJSON(char *theMemberName, char *channelName, char *root, int shouldCountMinusOne);`

Determines if the member is existent.

The function looks into the provided JSON string and searches for special JSONy signs, like double quotes, braces or brackets.

There is a second version of this function with the signature memberExistsInJSON_cJSON(char *theMemberName, char *channelName, cJSON *root, int shouldCountMinusOne), which uses the cJSON library to do the exact same tasks.

Parameters

— *theMemberName: The member's name.

— *channelName: The channel's name.

— *root: The JSON string the function should look into. The string represents the contents / parts of the contents of 'channels.txt'.

— shouldCountMinusOne: In the JSON saved file, parameters as "hasSeen": -1 are existent, indicating members who have left the channel. If this parameter is 1, then this members are also taken in account, and if it is 0, no.

Return Value

An integer indicating if the member exists (1) or not (0).

13. `int channelExistsInJSON(char *channelName, char *root);`

Determines if the channel is existent.

The function looks into the provided JSON string and searches for special JSONy signs, like double quotes, braces or brackets.

There is a second version of this function with the signature channelExistsInJSON_cJSON(char *channelName, cJSON *root), which uses the cJSON library to do the exact same tasks.

Parameters

— *channelName: The channel's name.

— *root: The JSON string the function should look into. The string represents the contents / parts of the contents of 'channels.txt'.

Return Value

An integer indicating if the channel exists (1) or not (0).

14. `int userExistsInJSON(char *user, char *root);`

Determines if the user is existent.

The function looks into the provided JSON string and searches for special JSONy signs, like double quotes, braces or brackets.

There is a second version of this function with the signature userExistsInJSON_cJSON(char *user, cJSON *root), which uses the cJSON library to do the exact same tasks.

Parameters

— *user: The user's username.

— *root: The JSON string the function should look into. The string represents the contents / parts of the contents of 'users.txt'.

Return Value

An integer indicating if the channel exists (1) or not (0).

15. `int userIDHavingGivenToken(char *token);`

Finds the ID of the user having the given token. If returned a valid ID, the user could be authenticated.

The function takes use of allUsers[] global array.

Parameters

— *token: The received token.

Return Value

An integer indicating the user's index in allUsers[] global array.

16. `void process(char *request);`

Processes the given request and calls the necessary Parsing Functions on demand.

The function takes use of splitString() to find the formatting of the received request from the client.

Parameters

— *request: The received request from the client.

Return Value

None

17. `void initializeAsciiArt(void);`

Fills the global asciiArt[] array of strings by the ASCII art pre-made for the word 'CGRAM'.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

None

Return Value

None

Network Functions

These functions have the role of connecting to the client, sending requests and receiving responses on demand.

1. `void chat(int client_socket, int server_socket);`

    Sends and receives data using the given socket identifiers.

    After zero-ing the request variable, this function attempts to receive new data from the client.

    Parameters

    — client_socket: The client socket's identifier.

    — server_socket: The server socket's identifier.

    Return Value

    None

2. `void connectServer(void);`

    Connects the server to the client.

    The function defines a sockaddr_in, then calls bind(), listen(), accept(), recv() and send() to achieve its goal.

    Parameters

    None

    Return Value

    None

Parsing Functions

These functions parse and process the requests and the responses and turn them into a recognizable format for other functions.

1. `void badRequest(char *result);`

    By copying data into *result, indirectly tells the client that the last request sent was using an unrecognizable format.

    The function uses the <string.h>-provided strcpy() function to achieve this functionality.

    Parameters

    — *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

2. `void doLogin(char *username, char *password, char *result);`

By copying data into *result, indirectly tells the client that the login request has been successful and also a token is available now (passed in the same parameter).

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *username: The user's username.

— *password: The user's password.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

3. `void doRegister(char *username, char *password, char *result);`

By copying data into *result, indirectly tells the client that the register request has been successful.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *username: The user's username.

— *password: The user's password.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

4. `void doLogout(char *token, char *result);`

By copying data into *result, indirectly tells the client that the logout request has been successful.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *token: The client's token.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

5. `void doCreateChannel(char *channelName, char *token, char *result);`

By copying data into *result, indirectly tells the client that the channel creation request has been successful.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *channelName: The channel name to create.

— *token: The client's token.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

6. `void doLeave(char *token, char *result);`

By copying data into *result, indirectly tells the client that the channel leaving request has been successful.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *token: The client's token.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

7. `void doJoinChannel(char *channelName, char *token, char *result);`

By copying data into *result, indirectly tells the client that the channel joining request has been successful.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *channelName: The channel name to join.

— *token: The client's token.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

8. `void doSearchBetweenMessages(char *textToSearch, char *token, char *result);`

By copying data into *result, indirectly tells the client that the text searching request has been successful.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *textToSearch: The text to be searched for.

— *token: The client's token.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

9. `void doRefresh(char *token, char *result);`

By copying data into *result, indirectly tells the client that the message reloading request has been successful.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *token: The client's token.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

10. `void doSend(char *originalMessage, char *token, char *result);`

By copying data into *result, indirectly tells the client that the message adding request has been successful.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *originalMessage: The message to be added to the current channel's messages.

— *token: The client's token.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

11. `void doSearchBetweenMembers(char *textToSearch, char *token, char *result);`

By copying data into *result, indirectly tells the client that the member searching request has been successful.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *textToSearch: The member's name to be searched for.

— *token: The client's token.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None

11. `void doChannelMembers(char *token, char *result);`

By copying data into *result, indirectly tells the client that the 'all members finding' request has been successful.

The function uses the <string.h>-provided strcpy() function to achieve this functionality.

Parameters

— *token: The client's token.

— *result: The result of the actions done in the function, which is then sent back to the client.

Return Value

None