Here is the translation of the text into English:

"This laboratory assignment is the most important examination in this course. If you can complete it independently, you have the knowledge needed for further studies (remember not to help each other with the code). It is demanding and will take time, but when you are done, you will be programmers. You have a guide at the end of this document to assist you. Using it is optional. The program should work according to this document but must also be well-programmed. This means, first and foremost, that it should be well-structured using functions and should use arrays and structs where appropriate. No global variables are allowed. Additionally, you should choose good variable and function names. Comment sparingly and focus on making the code itself easy to read. Note that this program should be written in a single source code file.

Task:

The task is to create a database of board game ratings by users. The program has two login options, as an administrator and as a user. As an administrator, you can add and remove users. As a user, you can add and remove board games and ratings. All users and their board games and ratings are saved in a text file when the program exits and can be read in when the program starts.

Data:

All users and all their board games and ratings should be saved in a text file so they can be read in when the program starts. The user specifies the file when the program starts. If the file doesn't exist, the program starts without any data. This means that you can only log in as an administrator (admin). A strong recommendation is that the program only reads from the file at the start and writes to the file only at the end. During execution, all changes to the data are made as it is stored in the program. You can choose how to organize the text file. It's good if it's reasonably human-readable, but the focus should be on easy readability by the program. To simplify reading, you can assume that all usernames and board game names consist of only one word.

Internally, I have the following suggestion for how to organize the data. If you want to organize the data differently, talk to your teacher and get it approved. Create a game struct that contains a game's name and a rating. You can assume that a game name is never longer than 40 characters and consists of one word. The rating is an integer from 1 to 10. Create a user struct that contains a username (also a maximum of 40 characters and only one word), an array with room for 20 game structs, and an integer that keeps track of the number of games the user has entered. Store all users in an array with room for 30 user structs. Use #define for all maximum limits so you

can easily set them to low values when you want to test how the program behaves when you try to enter more than what's allowed. You need to be able to handle a user trying to store more than 20 games and an admin trying to register more than 30 users. However, you don't need to handle someone entering a longer username or game name than 40 characters.

Start and Start Menu:

When the program starts, you must specify a text file (you can assume that the filename is not more than 40 characters). If the file exists, the program can assume it has the correct format and read the data. The program can assume that users fit in the array, and there are no more than 20 games per user in the file. When the program exits, it writes all users and their games and ratings to the text file that was specified when the program started (this also applies if the file didn't exist when the program started). After specifying the text file, you can enter a username, admin, or quit. If you enter a username, the program checks if such a user exists. If so, you enter the user menu until you choose Exit. If the user doesn't exist, you will be informed, and you get another chance to enter a choice. If you enter admin, you enter the admin menu until you choose Exit. If you enter quit, the program saves all data to the previously specified text file and exits. A run can look like the following (bold is user input):"

```
Welcome to boardgame ratings.
Which file do you want to use: ratings2.txt
Please enter user name, admin or quit: Peter
User does not exist
Please enter user name, admin or quit: Petra
User does not exist
Please enter user name, admin or quit: admin
Administration
          1) Add user
          2) Remove user
          3) Print all users
          4) Print all users and all their ratings
          5) Exit
Choose: 5
Please enter user name, admin or quit: Afshin
Afshin's boardgames
          1) Print games
          2) Add game
          3) Search games
          4) Remove game
          5) Exit
Choose: 5
Please enter user name, admin or quit: quit
Saving all data to ratings2.txt
Goodbye!
```

Administration:
Here, we will go through what the various options should do in the administration menu. When you finish an option, you should return to the administration menu (except if you choose Exit).

```
Administration
          1) Add user
          2) Remove user
          3) Print all users
          4) Print all users and all their ratings
          5) Exit
Choose:
```

Add User:
Here, you have the opportunity to add new users. When you enter a new username, the program checks that this user does not already exist and that there is space in the array. If the user already exists, the program informs you that the name is taken. If the array is full, the program notifies you of this. Otherwise, the new user is created, which means a new struct is added to the array with the new username and zero board games. Since users should be listed in alphabetical order, it's best to keep the list sorted by inserting each name in the right place in the list. When you want to finish, enter 'q', and you will return to the admin menu. When you choose Add user, it should look like the following:

```
Register new user (q to quit): Afshin
User name exist! Please choose another
Register new user (q to quit): Sara
User name exist! Please choose another
Register new user (q to quit): Samira
Register new user (q to quit): Peter
Register new user (q to quit): Noor
The register is full!
Register new user (q to quit): q
```

Remove User:
When you choose this option, you can unregister a user. This means that the user and all the board games and ratings they have registered will be removed. Internally, we don't want to create gaps in the array, so when we remove a user, we shift all the users after them in the array up by one step. If the selected user has registered board games, you should receive a warning and have the chance to reconsider before removing the user. When you choose Remove user, it should look like the following:

```
Remove user (q to quit): Afshin
Removed
Remove user (q to quit): Ella
Warning: User has rated games.
Do you still want to remove Ella (y/n)? n
Remove user (q to quit): Annan
User do not exist! Please choose another.
Remove user (q to quit): q
```

..

Print All Users:
When you choose this option, all users in the program will be printed in alphabetical order, and then the admin menu will reappear. It should look like the following:

```
Users:

_____________________________________________
Amritpal
Lubna
Mira
Peter
```

Or if there are no users:
“No users registered”

Print All Users and All Their Ratings:
When you choose this option, all users and their board games with ratings will be printed, and then you will return to the admin menu. Users are listed in alphabetical order, and the games are printed in the order they were added. It should look like the following

```
Users and boardgames:
_____________________________________
Amritpal
          Monopoly            5
          Risk                7
          Tic-tac-toe         6
Lubna
          Pictonary           4
          Acquire             7
          Monopoly            8
Mira
          Scrabble            10
          Catan               3
Peter
          No games registrered
Ralf
          Chess               8
          Checkers            9
```

User Menu:
Here, we will go through what the various options should do in the user menu. All of these options are specific to the logged-in user. When you finish an option, you should return to the user menu (except if you choose Exit).

```
Afshin's boardgames
          1) Print games
          2) Add game
          3) Search games
          4) Remove game
          5) Exit
Choose:
```

Print Games:
When you choose this option, the user's board games with ratings will be printed, and then you will return to the user menu. The games are printed in the order they were added by the user. This depends on how the user has added them. It should look like the following:

```
Monopoly            5
Risk                7
Tic-tac-toe         6
```

Add Game:
In this option, the user has the opportunity to add new games if there is space in the array. If there is no space, the program will notify the user and return to the user menu. When the user enters a game, the program checks that the game does not already exist. If the game already exists, the program informs the user. Otherwise, the user can enter a rating, and the game is added to the end of the user's list in the array. If the array becomes full, the program will notify the user and return to the user menu. The user can continue to add new games until they choose to quit. Ratings should be given as whole numbers from 1 to 10. If the user enters an invalid value, they will be given another chance until they enter a valid rating. When you choose Add game, it should look like the following:

```
Register new boardgame (q to quit): Risk
Boardgame already added.
Register new boardgame (q to quit): Monopoly
Boardgame already added.
Register new boardgame (q to quit): Power_grid
Rating (1-10): 11
Illegal value!
Rating (1-10): 0
Illegal value!
Rating (1-10): 10
Register new boardgame (q to quit): Checkers
Rating (1-10): 5
Your game register is full.
```

Search Games:
In this option, a user can search for their games using a text string. All games registered for the user that contain the search string will be displayed with their ratings. When you choose Search games, it should look like the following:

```
Search (q to quit): er
Power_grid          7
The_new_era         5
Terra_Mystica       2
Search (q to quit): q
```

And if you choose this option while there are no games, print “No games registered”

Remove Game:
In this option, the user is first given the opportunity to perform a search as shown in the example above. If the user doesn't get an exact match, they can search again until they find an exact match. Once the user finds an exact match, they can choose to remove that game. After that, they can continue to remove more games until they enter 'q' or the user's list of games is empty. It's important that the program is designed to use the same search code for this option and the search option above. This means that they should call the same search function, not copy the code. When removing a game, it's important not to create gaps in the array. Therefore, move all the games after the removed game up by one step. When you choose Remove game, it should look like the following:

```
Search boardgame to remove (q to quit): an
American_Rails      7
Canasta             6
You did not find one unique boardgame.
Search boardgame to remove (q to quit): American
American_Rails      7
Do you want to remove this game (y/n): y
Search boardgame to remove (q to quit): q
```

And if you choose this option while there are no games, print “No games registered”

Thats the whole task, and the teacher added this:

At the time of presentation, you should have a completed file with a minimum of ten users. Among these users, three should have at least ten games registered, and the remaining users should have at least three games registered. This will demonstrate a diverse set of user profiles with varying numbers of registered games.