Developer Documentation of Private blog console application

Users can sign up, log in, create, edit, and delete blog posts, as well as search for blog posts.. The program uses linked lists to store user and blog post information, and data is saved and loaded from files for persistence. There are two text files, one for users and another one for blog posts.

1.Data Structures

struct User
- It is a self-referential structure. Represents a user with elements such as name, email, contact information, username, password, and a pointer to the next user in the linked list.

struct BlogPost
- Represents a blog post with elements including date, author, time, keywords, content (dynamically allocated), and a pointer to the next blog post in the linked list.

2. Function List

1. void signUp(struct User** users)

Input Parameters: Pointer to the head of the user linked list.
- Result: Adds a new user to the list.

2. int logIn(struct User* users, struct User** currentUser)

- Input Parameters: User linked list and pointer to the current user.
- Result: Logs in a user and updates the current user pointer.

3. void subMenu(struct User* currentUser, struct BlogPost** blogs)

- Input Parameters: Current user and pointer to the head of the blog post linked list.
- Result: Displays and handles operations in the sub-menu.

4. void createBlogPost(struct User* currentUser, struct BlogPost** blogs)

- Input Parameters: Current user and pointer to the head of the blog post linked list.
- Result: Creates a new blog post and adds it to the list.

5. void editBlogPost(struct User* currentUser, struct BlogPost** blogs

Input Parameters: Current user and pointer to the head of the blog post linked list.
Result: Edits an existing blog post based on user input.

6. void deleteBlogPost(struct User* currentUser, struct BlogPost** blogs)

Input Parameters: Current user and pointer to the head of the blog post linked list.
- Result: Deletes an existing blog post based on user input.

7. void searchBlogPost(struct BlogPost* blogs)

Input Parameters: Pointer to the head of the blog post linked list.

Result: Displays blog posts based on user search criteria.

8. void printBlogPost(const struct BlogPost* post)

Input Parameters:Pointer to a blog post.
Result: Prints the details of a blog post.

9. void freeUserList(struct User* userList)

Input Parameters: Pointer to the head of the user linked list.
Result: Frees memory allocated for the user linked list.

10. void freeBlogList(struct BlogPost* blogList)

Input Parameters: Pointer to the head of the blog post linked list.
Result: Frees memory allocated for the blog post linked list.

11. void saveUsersToFile(const struct User* users)

Input Parameters: Pointer to the head of the user linked list.
Result: Saves user information to the "users.txt" file.

12. void saveBlogPostToFile(const struct BlogPost* post)

Input Parameters: Pointer to a blog post.
Result: Appends a blog post to the "blog_posts.txt" file.

13. void saveBlogsToFile(const struct BlogPost* blogs)

Input Parameters: Pointer to the head of the blog post linked list.
Result: Saves all blog posts to the "blog_posts.txt" file.

3. Dynamic memory allocation solution

- Maximum length for user and blog post attributes, defined as LENGTH=100
- But Memory allocation for content is dynamic and should be free. In order to get arbitrary length of content we should ask for length from the user. Then using malloc we can allocate memory for it.

4. File handling

With the help of file handling, we can store information about username, password and their personal information. Every structure has its own files, user and blogpost, it must save every information and behave like a database.