Step-by-step algorithm for the Personal Diary Application code:

1. Define the `struct DiaryEntry` to store username, date, and content for a diary entry.

2. Define the `struct User` to store username and hashed password for a registered user.

3. Declare function prototypes for various functions used in the program.

4. Define constants for the file names (`usersFileName` and `diaryFileName`) and the encryption key (`encryptionKey`).

5. Implement the `main()` function:

   a. Declare a variable `choice` to store the user's menu choice and `currentUser` to keep track of the currently logged-in user.

   b. Start an infinite loop to display the main menu and get the user's choice.

   c. If the user enters an invalid choice, display an error message and prompt for the choice again.

   d. Depending on the user's choice, call the corresponding functions (`registerUser()`, `loginUser()`, `displayUserMenu()`, or exit the program).

6. Implement the `displayMenu()` function:

   a. Display the main menu options to the user.

7. Implement the `displayUserMenu()` function:

   a. Display the user menu options to the currently logged-in user.

   b. Get the user's choice for the user menu.

   c. If the user enters an invalid choice, display an error message and prompt for the choice again.

   d. Depending on the user's choice, call the corresponding functions (`writeEntry()`, `viewEntries()`, `searchEntries()`, or log out).

8. Implement the `registerUser()` function:

   a. Create a `struct User` variable `user`.

   b. Open the users file in append mode.

   c. Get the username and password from the user.

   d. Hash the password using `simpleHash()` function and store it in the `user` variable.

e. Write the username and hashed password to the users file.

f. Close the users file.

g. Display a registration successful message.

9. Implement the `loginUser()` function:

a. Create a `struct User` variable `user`.

b. Open the users file in read mode.

c. Get the username and password from the user.

d. Hash the provided password using `simpleHash()` and store it in the `user` variable.

e. Read the username and hashed password from the users file, and compare it with the provided credentials.

f. If the credentials match, copy the username to the `currentUser` variable, close the users file, and return 1 (login successful).

g. If the credentials do not match, close the users file, and return 0 (login failed).

10. Implement the `writeEntry()` function:

a. Create a `struct DiaryEntry` variable `entry`.

b. Open the diary file in append mode.

c. Get the date and content of the entry from the user.

d. Validate the date format using `validateDate()` function, and prompt the user for a valid date if needed.

e. Encrypt the diary content using `encrypt()` function.

f. Copy the username to the `entry` variable.

g. Write the username, date, and encrypted content to the diary file.

h. Close the diary file.

i. Display an entry written successfully message.

11. Implement the `viewEntries()` function:

a. Create a `struct DiaryEntry` variable `entry`.

b. Open the diary file in read mode.

c. Print the header for the previous entries.

d. Read entries from the diary file.

e. If the username matches the `currentUser`, decrypt the content using `decrypt()` function, and display the date and content.

f. Close the diary file.

12. Implement the `searchEntries()` function:

a. Create a `struct DiaryEntry` variable `entry`.

b. Open the diary file in read mode.

c. Get the search term from the user.

d. Print the header for the search results.

e. Read entries from the diary file.

f. If the username matches the `currentUser`, decrypt the content using `decrypt()` function.

g. Check if the search term matches the date or content of the entry, and display the matching entries.

h. Close the diary file.

13. Implement the `encrypt()` function:

a. Iterate through the characters in the input text.

b. Shift each character by the `encryptionKey` positions to the right.

14. Implement the `decrypt()` function:

a. Iterate through the characters in the input text.

b. Shift each character by the `encryptionKey` positions to the left.

15. Implement the `validateDate()` function:

a. Parse the date string to extract day, month, and year using `sscanf()`.

b. Check if the parsed values are within valid ranges (day: 1-31, month: 1-12, year: 1900-2100).

c. If any of the conditions fail, return false; otherwise, return true.

16. Implement the `clearBuffer()` function:

    a. Clear any extra characters from the input buffer until a new line or EOF is encountered.