PASSWORD MANAGER – DOCUMENTATION

1. Overview

This project implements a **command-line password manager in C** that securely stores login credentials for websites or applications.
Passwords and related data are **encrypted before storage** and can be accessed **only after successful master password authentication**.

---

2. Key Features

- Master password protection
- Encrypted storage of credentials
- Add, view, and search credentials
- Optional random password generation
- File-based persistent storage

---

3. Data Structure Used

```
struct Credential {

    char site[50];

    char username[50];

    char password[50];

};
```

This structure groups related credential information, making file operations consistent and organized.

---

4. Files Used

File Name	Description
`master.txt`	Stores encrypted master password
`passwords.txt`	Stores encrypted credentials

---

5. Encryption Technique

A **Caesar cipher** is used where each character is shifted by **+3** during encryption and **−3** during decryption.

This approach is chosen for **simplicity and academic demonstration** of encryption concepts.

---

6. Module-Wise Explanation

6.1 Utility Module

Functions:

- `clearBuffer()`
- `inputString()`

Purpose:
Handles safe input and prevents input buffer issues caused by mixed usage of `scanf()` and `fgets()`.

6.2 Encryption Module

Functions:

- `encrypt()`
- `decrypt()`

Purpose:
Ensures sensitive data is not stored in plain text.
Encryption is applied before saving to files, and decryption is done only when displaying data.

6.3 Master Password Module

Functions:

- `setupMasterPassword()`
- `verifyMasterPassword()`

Purpose:
Controls access to the application.
If a master password does not exist, the user is prompted to create one.
Every program execution requires successful authentication.

6.4 Password Generator Module

Function:

- `generatePassword()`

Purpose:
Generates a random password using letters, digits, and symbols to avoid weak user-chosen passwords.

6.5 Credential Management Module

Functions:

- `addCredential()`
- `viewAll()`
- `searchCredential()`

Purpose:

- Add encrypted credentials to file
- Display all credentials after decryption
- Search credentials based on website/app name

Data is always **written and read in the same order** to avoid logical errors.

---

7. Program Flow

1. Verify master password
2. Display main menu
3. Perform selected operation
4. Repeat until exit

---

8. Limitations

- Basic encryption only (not production-grade)
- Password input not masked
- Local file storage
- No edit or delete functionality

---

9. Conclusion

This project demonstrates **secure data storage, file handling, structures, and basic encryption** in C.
It fulfills academic objectives while maintaining clean, modular, and understandable code.

---

10. Compilation & Execution

```
gcc password_manager.c -o password_manager
```

```
./password_manager
```

1️⃣ Utility Functions (Helper Functions)

These functions are reused throughout the program to avoid repetition.

`clearBuffer()`

What it does

Clears leftover input from the keyboard buffer.

Why it is needed

- `scanf()` leaves a newline (`\n`) in the buffer
- If not cleared, `fgets()` will read that newline instead of user input

How it works

```
while ((c = getchar()) != '\n' && c != EOF);
```

- Reads characters one by one
- Stops when newline or end-of-file is reached

👉 Prevents **skipped inputs and unexpected behavior**

`inputString(char *str, int size)`

What it does

Safely reads a string input from the user.

Why it is needed

- `scanf("%s")` breaks at spaces
- `gets()` is unsafe
- `fgets()` is safer but keeps `\n`

How it works

```
fgets(str, size, stdin);
str[strcspn(str, "\n")] = '\0';
```

- Reads input safely
- Removes the trailing newline

👉 Used for **all string inputs**

2️⃣ Encryption & Decryption Logic

`encrypt(char *str)`

What it does

Encrypts text using a **Caesar cipher**.

Logic

```
*str += 3;
```

- Each character's ASCII value is increased by 3
- Example:
  - a → d
  - 1 → 4

Why used

- Simple
- Easy to explain in viva
- Demonstrates encryption concept

`decrypt(char *str)`

What it does

Reverses the encryption.

Logic

*str -= 3;

- Exactly undoes what `encrypt()` did

👉 Encryption and decryption are **perfect inverses**

3️⃣ Master Password Handling

`setupMasterPassword()`

When it runs

- Only if `master.txt` does not exist

What it does

- Takes master password from user
- Encrypts it
- Stores it in a file

Why

- First-time setup
- Prevents hard-coding passwords

Key idea

> Master password is **never stored in plain text**

`verifyMasterPassword()`

What it does

Validates access before program starts.

Logic

1. If `master.txt` not found → setup password
2. Else:
    - Read encrypted password
    - Encrypt user input
    - Compare both

Why encrypt input instead of decrypting file

- Safer practice
- Stored password remains unchanged

👉 If passwords match → access granted
👉 Else → program exits

4️⃣ Password Generator

`generatePassword(char *pass, int length)`

What it does

Creates a random password automatically.

Characters used

- Uppercase letters
- Lowercase letters
- Digits
- Symbols

Logic

pass[i] = chars[rand() % (sizeof(chars) - 1)];

- Randomly picks characters from allowed set
- Ends with `\0` (null terminator)

Why useful

- Prevents weak passwords
- Optional feature, user-controlled

5️⃣ Core Application Functions

`addCredential()`

Purpose

Adds a new credential securely.

Steps

1. Ask for site name
2. Ask for username
3. Ask how password should be created
4. Encrypt all fields
5. Store them in file

Important detail

```
fprintf(fp, "%s %s %s\n", c.site, c.username, c.password);
```

👉 **Write order is fixed and consistent**
This fixes your earlier bug.

`viewAll()`

Purpose

Displays all stored credentials.

Logic

1. Read encrypted data from file
2. Decrypt each field
3. Display neatly

Key point

- Decryption happens **only in memory**
- File remains encrypted

`searchCredential()`

Purpose

Search credentials by site/app name.

Logic

1. Ask user for site
2. Read each record
3. Decrypt temporarily
4. Compare site names
5. Stop at first match

Why search by site

- Logical
- User-friendly
- Avoids ambiguity

6️⃣ Program Flow (Big Picture)

Overall Logic

1. Verify master password

2. Show menu
3. Perform selected action
4. Repeat until exit

Why this design

- Modular
- Easy to debug
- Easy to explain
- Easy to extend

7️⃣ Important Things You Should Remember (Viva Gold)

✔ File read order **must match write order**
✔ Encrypt before saving, decrypt only when displaying
✔ Master password protects everything
✔ Caesar cipher is for learning, not real security
✔ Structures group related data
✔ Utility functions prevent repetition

1️⃣ POSSIBLE VIVA QUESTIONS + ANSWERS

Q1. Why did you use a structure in this program?

Answer:
A structure is used to group related data such as website name, username, and password into a single unit. This makes the code more organized, readable, and easier to manage when storing or retrieving credentials from files.

Q2. Why did you use file handling instead of arrays?

Answer:
File handling allows data to persist even after the program terminates. Arrays store data only during runtime, whereas files enable long-term storage of credentials.

Q3. Why is `fgets()` preferred over `scanf()` for string input?

Answer:
`fgets()` is safer because it prevents buffer overflow and can read spaces. `scanf("%s")` stops at whitespace and may cause input-related bugs.

Q4. Why is `clearBuffer()` necessary?

Answer:
`scanf()` leaves a newline character in the input buffer. If it is not cleared, the next `fgets()` call may read that newline instead of user input, causing skipped inputs.

Q5. Why do you encrypt before saving and decrypt after reading?

Answer:
Encrypting before saving prevents passwords from being stored in plain text. Decrypting only when displaying ensures the stored data remains protected at rest.

Q6. What happens if someone opens the password file directly?

Answer:
They will only see encrypted text, which is not human-readable without the decryption logic and the master password.

Q7. Why did you use a master password?

Answer:
The master password ensures that only authorized users can access or view stored credentials. It acts as the first layer of authentication.

Q8. How does the password generator work?

Answer:
It randomly selects characters from a predefined set of uppercase letters, lowercase letters, digits, and symbols using the `rand()` function.

2️⃣ EXPLAIN THE BUG YOU ENCOUNTERED (TOPPER-STYLE)

💡 **This answer matters a LOT in viva.**

Question: *Did you face any bugs while developing this project?*

Answer (say exactly this):

> Yes. I encountered a logical bug related to file handling and data interpretation.
> Initially, the credentials were written to the file in one order, but when reading them back, they were interpreted in a different logical order.
> Because of this mismatch, the website name, username, and password appeared swapped during display and search operations.
> I fixed this by ensuring that the write format and read format were perfectly consistent and by clearing old data files that were written using the previous format.

👉 This shows:
✔ Debugging skill
✔ Understanding of file I/O
✔ No blaming compiler or OS

3️⃣ HOW TO JUSTIFY WEAK ENCRYPTION (VERY IMPORTANT)

⚠️ **Never say “because it’s simple” alone.**
Say this instead:

Question: *Why did you use such a weak encryption technique?*

Perfect answer:

> The purpose of this project is academic and focuses on demonstrating the concept of encryption rather than implementing production-level security.
> The Caesar cipher was chosen because it is easy to implement, easy to explain in a viva, and clearly demonstrates how encryption and decryption work.
> In real-world applications, stronger algorithms such as AES or hashing techniques would be used instead.

🔥 This answer shows:
✔ Awareness
✔ Maturity
✔ No overclaiming

4️⃣ RAPID-FIRE QUESTIONS (1-LINE ANSWERS)

Question	Answer

Why not store passwords in plain text?	It is insecure and exposes sensitive data
What is encryption?	Converting readable data into unreadable form
What is decryption?	Converting encrypted data back to original form
Why use `struct`?	To group related data
Why use files?	For persistent storage
Why random passwords?	To avoid weak user-chosen passwords