**Project Title:** Password Manager

**Objective:**

The objective of this project is to design and implement a secure and efficient Password Manager system using C++. The system will allow users to safely store, retrieve, and manage their passwords while ensuring data security and quick access through the effective use of data structures and algorithms. This project emphasizes the practical implementation of core Data Structures and Algorithms (DSA) concepts to create a real-world solution that promotes secure password management.

**1.Problem Description and Project Aims**

With the increasing number of online platforms, users must maintain multiple unique passwords for different accounts. Remembering all these passwords is often difficult, leading many individuals to reuse weak passwords or store them insecurely in plain text files. Such practices expose users to risks of identity theft and data breaches.

The **Password Manager** aims to solve this problem by providing a secure and efficient system for managing user passwords locally. The project will focus on the following objectives:

* Enable users to securely store, update, and delete passwords.
* Use efficient data structures for fast searching and retrieval.
* Implement encryption to protect stored data.
* Allow users to monitor and manage password strength.

This project highlights the use of **C++ data structures and algorithms** to create a real-world application emphasizing security, efficiency, and reliability.

**2. Key Data Structures**

To achieve the above objectives, the following data structures will be implemented:

1. **LinkedList:**  
   Used for dynamic storage of password records. Each node will store an account name, username, and encrypted password. This allows easy insertion and deletion of passwords without excessive memory usage.
2. **Hash Table (Map)**  
   Used for constant-time password lookup by mapping account names (keys) to passwords (values). This ensures efficient search and retrieval operations.
3. **Stack:**  
   Used to support undo operations (e.g., reverting a deleted or updated record). The last operation is stored on top of the stack and can be easily reversed.
4. **Queue:**  
   Used to maintain a log of recent actions performed by the user (e.g., login activity or password access history). It follows the FIFO (First-In, First-Out) principle to display actions in order.
5. **Priority Queue(Heap):**  
   Used to prioritize passwords based on their strength or frequency of use. For example, a max-heap can quickly identify the most frequently used or strongest passwords, while a min-heap can identify the weakest passwords that require updates.
6. **String and Character Arrays:**  
   Used to handle encryption, decryption, and password input efficiently.

**3. Main Algorithms**

1. **Hashing Algorithm:**  
   Used for mapping account names to hash values for quick retrieval in O(1) average time.
2. **Encryption and Decryption Algorithm:**  
   A lightweight algorithm such as the Caesar Cipher or XOR-based encryption will be used to secure stored passwords.
3. **Search Algorithm:**  
   Depending on the data structure, hashing or linear search will be used to locate passwords.
4. **Heap Operations:**  
   Implemented for the priority queue to maintain and retrieve passwords based on priority (e.g., weakest or strongest).
5. **Queue Operations:**  
   Used to enqueue recent activities and dequeue old ones to maintain a manageable history.
6. **File Handling Algorithm:**  
   Handles reading and writing of encrypted passwords to and from a local file, ensuring data persistence between sessions.

**4. Data Flow: Receiving, Processing, and Displaying Data**

**Receiving Data:**  
Users will input their credentials (account name, username, and password) through a console-based interface.

**Processing Data:**  
Upon receiving input, the system will:

* Encrypt the password for security.
* Store it in the **hash table** or **linked list** for dynamic and efficient management.
* Use the **priority queue** to rank passwords based on strength.
* Record the action in the **queue** for activity tracking.

**Displaying Data:**  
When a user requests to view or search for a password, the system will decrypt and display it. The queue ensures recent actions are viewable, while the priority queue helps display passwords according to their security level.

**5. Integration of Key DSA Concepts**

This project integrates the following DSA concepts:

* **Dynamic Memory Allocation:** Implemented through linked lists for flexible record management.
* **Hashing:** Ensures efficient search and retrieval of password data.
* **Stack and Queue Operations:** Demonstrates LIFO and FIFO principles through undo and activity tracking features.
* **Priority Queue (Heap):** Shows advanced data structure use for ranking or prioritizing records.
* **File Handling:** Combines DSA with persistent data storage techniques.
* **Encryption Algorithms:** Integrates algorithmic logic for security enhancement.

By combining these data structures and algorithms, the project demonstrates a complete and practical application of DSA principles within a real-world problem domain.

**Conclusion**

The proposed **Password Manager** project applies the principles of Data Structures and Algorithms to develop a secure, efficient, and reliable system for password storage and management. The integration of **linked lists, hash tables, stacks, queues, and priority queues** ensures optimal performance in terms of data organization, retrieval speed, and security. By implementing encryption and file handling, the project bridges theoretical DSA knowledge with practical cybersecurity applications, providing a strong foundation for efficient password management in real-world systems.