Password Manager in C++

Course: Data Structures and Algorithms

Submitted by: Ahmed Ali khan (2024063) , Muhammad Abdullah Kaleem (2024326), Muhammad Naqi Afaq (2024433)

Instructor: Mr. .Zubair Ahmed

Date: October 2025

# 1. Objective

The objective of this program is to design a simple Password Manager application in C++ using the concept of linked lists and user authentication. The program allows users to securely log in, add passwords for various accounts, check password strength, and view stored passwords only after verifying authentication.

# 2. Program Overview

This program demonstrates the use of linked lists, structures, and basic authentication mechanisms. It simulates a password manager where the user can log in, add, and view stored passwords. The system ensures password protection by requiring re-authentication before viewing stored passwords.

# 3. Key Features

• User authentication with email and password

• Linked list implementation for password storage

• Password strength checking with recommendations

• Secure viewing of stored passwords after re-verification

• Option to log out and clear all stored passwords

# 4. Code Explanation

## 4.1 Structures Used

Two structures are defined in this program:  
1. PasswordNode – stores an account name and its password, forming a linked list.  
2. UserAuth – stores the current user’s email and password for authentication purposes.

## 4.2 User Authentication

The `authenticateUser()` function prompts the user for their email and password. If both inputs are non-empty, the login is considered successful. The entered credentials are stored in a global `UserAuth` pointer for later verification.

## 4.3 Password Verification

The `verifyPassword()` function ensures that only the logged-in user can view stored passwords. It asks for the password again and compares it with the one stored in `currentUser`. Only if they match, access is granted.

## 4.4 Password Strength Checker

The function `checkAndSuggestStrength()` analyzes a user-entered password and suggests improvements if it lacks uppercase letters, digits, or special symbols. This encourages users to create stronger passwords.

## 4.5 Linked List Password Storage

The structure `PasswordManager` maintains a singly linked list of passwords. Each node represents an account. The function `addPassword()` adds new accounts, while `viewPasswords()` displays all stored credentials. The list is dynamically managed and cleared upon logout to ensure data privacy.

## 4.6 Logout and Cleanup

The `logout()` function confirms if the user wants to log out. If confirmed, all stored passwords are deleted, and the user is logged out. Memory is freed to prevent leaks.

# 5. Program Flow

1. Program starts and prompts user to log in.  
2. After successful login, the main menu is displayed with the following options:  
 - Add Password  
 - View All Passwords  
 - Logout  
 - Exit  
3. If the user selects Add Password, they can store new credentials.  
4. Viewing passwords requires entering the login password again.  
5. Logout clears all saved data, ensuring privacy.  
6. The program terminates when the user selects Exit.

# 6. Example Execution

========== LOGIN REQUIRED ==========  
Enter your email: user@example.com  
Enter your password: 12345  
✅ Login successful! Welcome to Password Manager.  
  
====== PASSWORD MANAGER ======  
1. Add Password  
2. View All Passwords  
3. Logout  
4. Exit  
  
Enter your choice: 1  
Enter Account Name: Gmail  
Enter Password: MyPass123!  
✅ Password for Gmail added successfully!

# 7. Conclusion

This project demonstrates practical implementation of linked lists, structures, and authentication concepts in C++. It enhances understanding of dynamic memory allocation and data handling. The Password Manager provides a simple yet effective example of user data protection, verification, and password strength analysis.