**Password Management Program Documentation**

**Overview**

This C program allows users to set, update, and compare passwords. It uses ANSI escape codes to provide color-coded feedback in the terminal and includes functionalities for password length comparison and concatenation.

**Features**

* Set a password
* Update the password
* Compare the entered password with the saved password
* Display length differences between passwords
* Color-coded feedback for user actions

**1. Research**

**Objective**

To create a simple password management tool that allows users to set and verify passwords securely.

**Background**

Password management tools are essential for ensuring digital security. This program serves as a basic illustration of password handling in C, utilizing standard libraries and features.

**Existing Solutions**

Numerous libraries and applications exist for password management, including:

* **Password managers** like LastPass and Bitwarden, which offer extensive features but are often overkill for basic needs.
* **Simple scripts** that handle passwords in various languages, but often lack user-friendly interfaces.

**2. Analysis**

**Requirements**

* Input and output through the console.
* String manipulation for password handling.
* ANSI escape codes for visual feedback.

**Constraints**

* Limited to the capabilities of C language.
* Basic error handling for inputs.

**Use Cases**

1. User sets a password.
2. User compares the password with an input password.
3. User updates the password.

**3. Ideate**

**Concepts**

* Using fgets for input to handle spaces and special characters.
* Using strcspn to remove trailing newline characters from input.
* Using strcmp to compare passwords.

**User Interface**

The command-line interface will provide clear instructions and feedback using color codes.

**4. Build**

**Code Structure**

The program consists of:

* Functions for updating the password (updatePassword).
* Functions for comparing the password (comparePasswords).
* The main function for user interaction.

**Key Functions**

* **updatePassword(char \*savedPassword)**: Prompts the user to enter a new password.
* **comparePasswords(const char \*savedPassword)**: Compares the saved password with user input and displays the result.

**Example Usage**

c

Copy code

int main() {

// Password management logic

}

**5. Test**

**Testing Strategy**

1. **Unit Tests**: Test each function independently (e.g., ensure updatePassword correctly updates and formats the password).
2. **Integration Tests**: Test the interaction between functions, especially between updating and comparing passwords.
3. **User Acceptance Testing**: Ensure the user experience is smooth and feedback is clear.

**Test Cases**

1. Enter a password and check its length.
2. Compare correct and incorrect passwords.
3. Update the password and verify the update.

**6. Implementation**

**Deployment**

* Compile the C program using a standard compiler (e.g., gcc).
* Run the executable in a terminal that supports ANSI codes.

**Example Compile Command**

bash

Copy code

gcc password\_manager.c -o password\_manager

Git Hub Link – https://github.com/PracticalTeam077/Assigment-4/tree/main

**Running the Program**

bash

Copy code

./password\_manager

**7. Maintenance**

**Ongoing Support**

* Monitor for bugs or issues reported by users.
* Consider user feedback for improvements, such as adding password strength validation or more advanced features