C++ Password Generator

Introduction

This document unveils the intricacies of the C++ code powering a robust password generator application. This program equips users with the ability to craft secure, random passwords tailored to their specific requirements. It boasts flexibility, allowing users to define the desired password length and incorporate a variety of character types, encompassing uppercase and lowercase letters, numbers, and symbols.

Functionalities

1. **Password Generation:**
   At the heart of the application lies the `generatePass` function, responsible for orchestrating the password generation process. It accepts five arguments:
   - `length`: The intended length of the password.
   - `includeUppercase`: A boolean flag indicating the inclusion of uppercase letters (True for inclusion, False for exclusion).
   - `includeLowercase`: A boolean flag indicating the inclusion of lowercase letters.
   - `includeNumbers`: A boolean flag indicating the inclusion of numbers.
   - `includeSymbols`: A boolean flag indicating the inclusion of symbols.

   The `generatePass` function meticulously constructs the password through the following steps:
   6. It meticulously initializes separate vectors to serve as repositories for characters of distinct types (uppercase, lowercase, numbers, and symbols).
   7. The function meticulously populates these vectors based on the designated ASCII character range for each type.
   8. A new vector named `pickFrom` is meticulously crafted to house characters eligible for password generation based on user selections (uppercase, lowercase, numbers, symbols). Characters are meticulously added to `pickFrom` based on the corresponding boolean flags.

9. To ensure an element of surprise, the function leverages the Fisher-Yates shuffle algorithm, implemented within the `shuffleArray` function (explained later). This algorithm meticulously shuffles two vectors:
    - `lengthIndices`: A vector containing indices representing the password length (utilized for random selection within `pickFrom`).
    - `pickFrom`: The vector holding characters eligible for password generation.
10. The function meticulously iterates `length` times, simulating the password creation process. In each iteration:
    - The shuffled `lengthIndices` vector meticulously provides a random index to select a character from the shuffled `pickFrom` vector.
    - The meticulously chosen character is appended to the `password` vector, effectively building the password character by character.
11. Finally, the function triumphantly returns the generated password as a character vector.

2. **User Interaction:**
   The `main` function serves as the program's entry point, orchestrating the user interaction.
    - It extends a warm welcome to the user with a centered title displayed using `setw` from `<iomanip>`.
    - The function prompts the user to enter the desired password length using `cin` from `<iostream>`.
    - It then inquires about user preferences regarding character types through a series of `cout` and `cin` interactions. Each prompt seeks a yes/no response (1 for inclusion, 0 for exclusion) for uppercase, lowercase, numbers, and symbols.
    - The program performs a crucial validation check to guarantee at least one character type is selected for password generation. If all boolean flags are `false`, an error message is displayed, and the program gracefully exits using `exit(1)`.

- Once valid user input is received, the `main` function calls upon the `generatePass` function, passing the gathered information (length and character type preferences).
- The generated password, retrieved from the `generatePass` function, is proudly displayed to the user.
- The program meticulously calculates the total number of possible password combinations based on the selected character types and password length using `pow` from `<cmath>`. This calculation considers the number of characters available in each included character type raised to the power of the password length.
- The program informs the user about the number of possible password combinations, emphasizing the importance of securing the generated password.
- Finally, a farewell message is displayed using `cout` from `<iostream>`.

3. **Shuffling Functionality:**
   The `shuffleArray(vector<char> &arr)` and `shuffleArray(vector<int> &arr)` functions (overloaded) implement the Fisher-Yates shuffle algorithm, a cornerstone of randomization techniques, to meticulously shuffle the elements within an array. The function is overloaded to work with both character and integer vectors.
   The `shuffleArray` function meticulously performs the following actions:
   - It accepts a reference to the vector (`arr`) that necessitates shuffling as input.
   - The function meticulously retrieves the size of the vector (`n`).
   - It meticulously utilizes `srand(time(0))` from `<cstdlib>` to meticulously seed the random number generator with the current time, guaranteeing a distinct randomization sequence for each program execution.
   - The core shuffling logic meticulously iterates from n-1 down

Conclusion:

This C++ password generator application offers a valuable tool for users seeking to create secure and random passwords. The program's flexibility allows for customization based on individual needs, and the clear user interaction guides users through the password generation process. Here are some potential areas for future enhancements:

- **Input Validation:** Strengthening error handling for invalid user input (e.g., negative password length).
- **Password Strength Measurement:** Providing feedback on the generated password's strength based on length, character types, and complexity.
- **Integration with Password Managers:** Exploring the possibility of integrating with password management tools for secure password storage.

By incorporating these enhancements, the application can further empower users to create and manage strong passwords effectively.