**Password Generator Program**

**Module title:Object Oriented Programming**

**Module code:IY101**

**Assessment name:**

**Chosen Title:**

**Student ID**:439981

***I confirm that this assignment is my own work.***

***Where I have referred to academic sources, I have provided in-text citations and included the sources in the final reference list.***

A blue text on a white background

Description automatically generated

**KAPLAN INTERNATIONAL COLLEGE LONDON – IYO COMPUTER SCIENCE PROGRAMME**

**1.Introduction**

My C++ password generator program is designed to create strong and secure passwords by adhering to specific criteria. The central function, `gen\_Password()`, takes charge of generating passwords with randomized lengths, ensuring the inclusion of alphanumeric characters. A secondary function, `isValidPassword()`, validates the generated passwords based on predefined rules. This report will delve into the functionality of the code, highlight its limitations, and propose improvements for a more robust implementation.

**2. Functionality of `gen\_Password**

**Randomized Password Generation**

The core of the password generation process involves determining a random length for each password, ranging from 10 to 16 characters. This introduces an element of unpredictability, contributing to the overall security of the generated passwords.

**Character Type Inclusion**

The function strategically selects positions for lowercase, digit, and uppercase characters, guaranteeing a diverse set of character types in each password. This aligns with the specified criteria of having at least one digit, one uppercase letter, and one lowercase letter in the password.

**Avoidance of Consecutive Identical Characters**

To prevent consecutive identical characters, the function employs a loop with careful character selection logic. This ensures that no two consecutive characters are the same, irrespective of case.

**Validation with `isValidPassword()`**

The `main()` function tests the `gen\_Password()` function by generating 20 passwords and printing them to the console. The `isValidPassword()` function is then employed to validate each generated password, checking for adherence to the specified rules. The results of the validation process are displayed, providing insights into the reliability of the password generation functionality.

**3.Limitations**

**Predictability of Random Seed**

My program relies on seeding the random number generator with the current time, introducing an element of unpredictability. However, in certain scenarios, the predictability of the seed might be a limitation, affecting the overall randomness of the generated passwords.

**Algorithmic Complexity**

My algorithm employed to avoid consecutive identical characters could be further refined. The current approach may introduce complexity, and there is room for improvement to enhance efficiency and maintainability.

**Potential Patterns**

While my code attempts to avoid guaranteed patterns in generated passwords, there is a possibility that natural patterns may occur. Ensuring that patterns do not occur consistently in every generated password is crucial.

**4.Potential Improvements**

**Cryptographically Secure Random Number Generation**

To address predictability concerns, consider transitioning to cryptographically secure random number generation techniques. Utilizing libraries such as `<random>` could provide more secure and less predictable random numbers.

**Enhanced Algorithm for Character Selection**

A more sophisticated algorithm for character selection could be implemented to further improve the diversity of the generated passwords. This could involve a weighted selection based on the frequency of character types, ensuring a more even distribution.

**Pattern Avoidance Refinement**

Continued efforts can be made to refine the pattern avoidance mechanism, ensuring that even natural patterns are less likely to occur consistently in generated passwords.

**5.Conclusion**

In conclusion, my C++ password generator program effectively generates passwords that meet the specified criteria. This report has provided insights into the workings of my code, highlighted its limitations, and suggested potential improvements for future iterations. By addressing predictability concerns, refining the algorithm, and enhancing pattern avoidance, my program can further elevate the security and reliability of the generated passwords. The report serves as a roadmap for ongoing development and optimization efforts.

My code:





