POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Menadżer haseł

1. Cel projektu

Celem projektu jest wykazanie, że student nie tylko opanował podstawowe elementy języka C++, ale również ogólnie pojęte umiejętności tworzenia mniejszych projektów programistycznych, na co składają się między innymi umiejętności odpowiedniego doboru narzędzi oraz sporządzanie dokumentacji stworzonych modułów pomocniczych.

Celem projektu nie jest znalezienie i zainstalowanie bibliotek programistycznych.

2. Opis projektu

Projekt nie powinniśmy stosować tego samego hasła do różnych portali. Wyciek jednego z nich naraża nas na utratę danych w wielu miejscach. W związku z tym, aby zminimalizować to niebezpieczeństwo powinniśmy stosować 2FA (które nie jest częścią tego projektu) oraz różne hasła. W tym drugim mogą nam pomóc programy typu Password Manager takie jak na przykład KeePass.

Projekt polega na stworzeniu aplikacji konsolowej służącej do modyfikowania i czytania z pliku, w którym będą przechowywane hasła wraz z dodatkowymi informacjami dotyczącymi różnych portali. Będą one zaszyfrowane pojedynczym hasłem głównym.

Aplikacja winna być uruchamiana z poziomu linii poleceń (ang. *command line*). Bazując na komendach czytanych z klawiatury, wykonywać odpowiednie funkcje i zwracać interesujące nas informacje.

3. Opis działania

Użytkownik po uruchomieniu programu ma możliwość wyboru pliku, z którego mają zostać pobrane hasła i informacje. Następnie wpisuje hasło. Niezależnie od tego, czy wprowadzone hasło jest poprawne, czy też nie, program powinien odszyfrować na jego podstawie zawarte w pliku informacje. (Przy podaniu hasła nieprawidłowego odszyfrowane dane również będą niepoprawne).

Po odszyfrowaniu pliku użytkowni otrzymuje timestamp ostatniego odszyfrowania oraz listę możliwych czynności:

1. Wyszukaj hasła;
2. Posortuj hasła;
3. Dodaj hasło;
4. Edytuj hasło;
5. Usuń hasło;
6. Dodaj kategorię;
7. Usuń kategorię.

4. Wymagania niefunkcjonalne

Projekt powinien być zaimplementowany zgodnie z zasadami produkcji czystego kodu wysokiej jakości oraz zgodny z dobrymi praktykami programistycznymi (zarówno uniwersalnymi jak i stricte dotyczącymi języka C++). Mając na uwadze fakt, że część z tych kryteriów może być kwestią dyskusyjną, student winien upewnić się, jego zrozumienie danych pojęć pokrywa się z elementami przedstawionymi na ćwiczeniach, na przykład poprzez konsultacje z prowadzącym.

Implementacja winna być przemyślana. Oznacza to, że należy dobrać możliwie jak najlepsze narzędzia programistyczne i zastosować je wraz z dobrymi praktykami ich używania. Wynika z tego fakt, że rozwiązanie, które "*po prostu działa*", może nie zdobyć wymaganej do zaliczenia liczby punktów.

Niekompilujące się programy będą automatycznie otrzymywały zero punktów. Należy odpowiednio skorzystać z podzielenia projektu na wiele plików.

Projekt winien być dokumentowany. Wymaganym jest, aby każda nietrywialna funkcja, metoda oraz klasa (ale już nie atrybuty klas) miały dotyczący ich komentarz zgodny ze standardem Doxygen. Głównymi aspektami, na których należy się skupić, są parametry (`@param`), wartości zwracane (`@return`) oraz krótki opis działania (przeznaczenia) danej metody / klasy / funkcji.

W tym opisie nie należy zagłębiać się w szczegóły jak dana idea jest implementowana. Proszę się skoncentrować na opisywaniu tego **co** jest robione, a nie **jak** coś jest robione.

Dobrym przykładem jest chociażby dokumentacja klasy String z Javy, której opis metod można znaleźć tutaj.

W narzędziach typu CLion czy Visual Studio, po zaimplementowaniu funkcji, klasy czy metody, jesteśmy w stanie wygenerować szablon takiej dokumentacji za pomocą wprowadzania następującej sekwencji znaków tuż nad elementem, który chcemy udokumentować: `/**`, a następnie klikając przycisk `Enter`.

5. Wymagania funkcjonalne

Po uruchomieniu programu użytkownik ma możliwość wybrania jednego z plików znajdujących się w folderze programu lub ma możliwość podania bezpośredniej, absolutnej ścieżki do pliku.

Dane w pliku są zaszyfrowane. Sposób szyfrowania musi być autorski i w pełni rozumiany. Hasło ma stanowić nieodzowną cześć procesu szyfrowania i odszyfrowania. Trudność złamania takiego szyfru nie będzie oceniana, ale otwarcie pliku w edytorze tekstu i proste metody deduktcyjne nie powinny być wystarczające do odszyfrowania jego zawartości. To samo się tyczy modyfikacji tej zawartości.

Każda próba odszyfrowania pliku powinna zapisać timestamp takiej próby. Z uwagi na to, że sam plik nie przechowuje nigdzie informacji o poprawnym haśle, a timestamp musi być zawsze zmieniany (zarówno podczas "nieudanej" próby otwarcia i modyfikacji pliku) to timestamp będzie jedyną informacją zapisaną jawnie w całym pliku. W związku z tym musimy znaleźć inny sposób na jego ukrycie.

Jedną z możliwości byłoby rozłożyć tę informacje na różne linijki. Początkiem linii 11. mogłoby być `hhDDDD`, linii 22. `mmDDDD`, a 33. `ssDDDD`, gdzie `hh` to godzina `mm` to minuta a `ss` to sekunda ostatniego odszyfrowania. `DDDD` to dalsze zaszyfrowane dane, niemiejące nic wspólnego z samym timestampem.

Każde hasło musi zawierać co najmniej:

- Nazwę (Nazwa własna tego wpisu np. ”Hasło do Konta1 na Google”);
- Tekst reprezentujący samo hasło;
- Kategorię.

Każde hasło dodatkowo może zawierać:

- Strona WWW/ Serwis;
- Login.

Implementacja tego elementu jest wymagana, choć nie każde hasło musi zawierać te składowe. Mają one być opcjonalne.

Opis komend:

1. Wyszukaj hasła – zwraca hasła, które zawierają konkretne parametry.
2. Posortuj hasła – zwraca posortowaną listę wszystkich haseł. Ma umożliwiać posortowanie po co najmniej 2 różnych parametrach w tym samym czasie, czyli na przykład po nazwie i kategorii.
3. Dodaj hasało – dodaje nowe hasło do zaszyfrowanego pliku. Powinniśmy tu dać użytkownikowi możliwość wpisania własnego hasła i poinformować go na ile jest to bezpieczne hasło i czy nie zostało już wcześniej wykorzystane. Dodatkowo należy

zaproponować mu hasło automatycznie wygenerowane dając mu jednocześnie możliwość wybrania pewnych parametrów takich jak:
   a. Ilość znaków;
   b. Czy ma zawierać wielkie i małe litery;
   c. Czy ma zawierać znaki specjalne.
4. Edytuj hasło – pozwala na edycje danych w istniejącym już haśle.
5. Usuń hasło – usuwa wybrane hasło lub hasła. Przed każdym usunięciem powinniśmy powiadomić o tym użytkownika szczególnie jeżeli usuwane jest więcej niż jedno hasło.
6. Dodaj kategorie – dodaje nową kategorię, którą będziemy mogli wykorzystywać przy tworzeniu nowych haseł .
7. Usuń kategorie – usuwa kategorie wraz ze wszystkimi hasłami, które do tej kategorii są przypisane.

6. Kryteria oceniania

Tabela 1 określa liczbę punktów do zdobycia za każde z wymienionych wymagań.

Wymaganie	Liczba punktów	Dodatkowy komentarz
Odpowiednie dobieranie i wykorzystywanie narzędzi programistycznych. Poprawne rozdzielenie kodu aplikacji na wiele niezależnych od siebie modułów.	5	Przykładowo używanie standardowych algorytmów zgodnie z ich przeznaczeniem, używanie szablonów zamiast makr, odpowiednie dobieranie kontenerów zgodnie z ich przeznaczeniem. Podział implementacji na wiele plików, stworzenie odpowiednich przestrzeni nazw czy klas. Oczywiście w ramach rozsądku – nic na siłę.
Stosowanie się do *const-correctness*.	1	Oznacza to stosowanie `const` nie tylko wszędzie tam, gdzie można, ale też wszędzie tam, gdzie koncepcyjnie dany element nie powinien być zmieniany.
Unikanie niepotrzebnych kopii danych w programie.	1	Zgodnie z wiedzą przedstawioną na ćwiczeniach. Tyczy się to typów większych niż prymitywy – wskazanym jest kopiowanie zmiennych typu `int` czy `double` zamiast przekazywać je przez `const&` (oczywiście w przypadku niemodyfikowanych kopii oznaczenie ich przez `const` jest wciąż wymagane).
Sporządzenie dokumentacji zgodnej z uproszczonym standardem Doxygen.	4	

Klarowność komunikatów błędów przy niepoprawnym użytkowaniu aplikacji.	3	
Poprawnie działające szyfrowanie i odszyfrowanie pliku.	7	
Poprawnie działająca obsługa timestampów.	5	
Poprawna implementacja komendy nr 1.	2	
Poprawna implementacja komendy nr 2.	3	
Poprawna implementacja komendy nr 3.	5	
Poprawna implementacja komendy nr 4.	2	
Poprawna implementacja komendy nr 5.	2	
Poprawna implementacja komendy nr 6.	1	
Poprawna implementacja komendy nr 7.	3	
Przejrzystość menu i łatwość pracy z programem.	6	Program ma charakteryzować się łatwą i intuicyjną strukturą nawigacji, jasnym przekazem informacji i przejrzystą strukturą treści.

Tabela 1. Kryteria oceniania.

7. Obrona

Student będzie przepytany ze szczegółów implementacyjnych swojego rozwiązania. Brak zrozumienia i identyfikacji dowolnego fragmentu kodu może być podstawą do **niezaliczenia całego projektu**. W związku z tym sugeruje się, aby studenci, którzy ukończyli projekt semestralny długo przed terminem obrony, zadbali o to, aby wszystkie szczegóły projektu zostały przypomniane.

8. Dodatkowe uwagi

Zabrania się stosowania rozwiązań, których student nie rozumie w dobrym stopniu. Każde wykorzystane narzędzie musi być opanowane przez studenta. Zezwala się na używanie elementów, które nie były przedstawione na ćwiczeniach czy wykładzie, lecz należy równocześnie pamiętać, że odpytywanie podczas obrony może się skoncentrować również i na tych elementach.

W przypadku dowolnych niejasności należy skonsultować treść i wymagania projektu z prowadzącym ćwiczenia.