POLISH-JAPANESE ACADEMY OF INFORMATION TECHNOLOGY

Password manager

1. Project goal

The goal of this project is to demonstrate student's knowledge of basic elements of the C++ programming language as well as the general knowledge of creating smaller programming projects. Including, among other things, the proper selection of programming tools and creation of a documentation of constructed auxiliary modules.

The purpose of the project is not finding and installing third party libraries.

2. Project description

We should never use the same password for multiple sites. Sensitive data leak from just one of them puts us in danger of losing data from other sites. Thus, to eliminate this danger, we should always use 2FA (unrelated to this project) combined with many different passwords. Password managers, such as KeePass, help us with the latter.

The project is focused on creating a console application aimed at modifying and reading a file containing passwords associated with additional information regarding various websites. This data will be encrypted with the main password.

The application should be run using the standard command line. Based on the user's input it should execute given commands and display the returned information.

3. Functionality description

While running the program, the user should be able to choose a source file for passwords and associated data. Then, they input the main password. Regardless of its correctness, the program should decrypt the file (based solely on the inputted password). If the inputted password was incorrect, the decrypted data should appear incorrectly.

After decrypting the file, the user should be prompted with the timestamp of the most recent decryption of the opened file and the following list of further actions:

1. Search for passwords;

2. Sort passwords;
3. Add a password;
4. Edit a password;
5. Remove a password;
6. Add a category;
7. Remove a category.

4. Non-functional requirements

Project should be implemented in accordance with clean code production rules and good programming practices (both universal and specific to C++).

Bearing in mind that some of these criterium might be a moot point, student should make sure that their understanding of these concepts is compliant with elements presented in class – for example via consultation with the tutor.

The implementation should be thought trough. This means that student should choose the possibly best programming tools and use them according to the previously stated good practices. A solution that "just works" might not be sufficient to earn a passing grade.

Programs that do not compile will be granted 0 points. Student should make good use of splitting the project into multiple files.

The project should be documented. It is required that every non-trivial function, method and class (but not their attributes) has a comment concerning them compliant to the Doxygen standard. Main aspects that should be brought up are the parameters (`@param`), return values (`@return`) and a short description of functionality (purpose) of the method/function/class.

This description should not include details concerning the way it was implemented. Please put emphasis on **what** is done and not on **how** it is done.

A good example of this is the Java String class documentation that can be found here.

In tools like CLion or Visual Studio, after implementing the function, class, or method, one is able to generate a template of such documentation by inputting the following sequence of characters right above the element one wants to document: /**, followed by the `Enter` button.

5. Functional requirements

After running the program, the user should be able to choose the source file from the list of files present in the execution directory or choose a different file by inputting an absolute path.

Data in the source file are encrypted. The encryption algorithm should be invented by the student and very well understood. A main password is required for the encryption and decryption process. Safety of such algorithm (and how hard it is to break it) will not be graded but opening the source file in text edit accompanied with simple deduction methods should not be sufficient to manually decrypt the data. The same holds true for meaningful modifications of that data.

Each attempt do decrypt the file should be saved as a timestamp. Because the source file does not store any data about the correct main password and because the decrypting timestamp should always be marked (despite the attempt being successful or not), that timestamp should be the only non-encrypted information stored in that file. Because of that we should identify a different way of hiding it.

One of the many possibilities is to divide this information over multiple lines. 11th line could represent `hhDDDD`, 22nd line could represent `mmDDDD` and 33rd could represent `ssDDDD`, where `hh` represents the hour, `mm` the minute and `ss` the second of the last decryption attempt. `DDDD` is some sample data unrelated to the timestamp.

Each password must consist of at least:

- Name (Name of the entry, e.g.: ”Password for Google account”);
- Text representing the password;
- Category.

Each password may additionally include:

- URL;
- Login.

Implementation of those two above elements is mandatory. However, not all password entries are required to contain them. They are optional.

Commands description:

1. Search for passwords – returns passwords matching the specified criteria.
2. Sort passwords – returns a list of all passwords, sorted. It should be able to sort according to at least 2 parameters simultaneously, for example according to the name and the category.
3. Add a password – adds a new password to the encrypted file. The user should be able to specify their own password and be prompted with the information regarding that password's safety and uniqueness. Additionally, they should be prompted with an option to choose a randomly generated password that meets the requirements specified by the user, such as:
    a. Number of characters;
    b. Should it contain both uppercase and lowercase characters;
    c. Should it contain special characters.
4. Edit a password – enables the user to edit data on an already existing password.
5. Remove a password – removes the selected password(s). Password removal should require a confirmation from the user, especially is multiple passwords are to be removed.
6. Add a category – adds a new category to be used with creating new passwords.
7. Removes a category – removes a category and all the passwords associated with it.

6. Grading criteria

Table 1 specifies the number of points available for implementing a specific functionality and following the specific criteria.

Requirement	Number of points	Clarification
Appropriate selection and usage of programming tools. Appropriately splitting code into multiple independent modules.	5	E.g., using standard algorithms as they were intended to be used; using templates instead of macros; using the best container for a given task. Splitting implementation into multiple files, classes, and namespaces. Of course one shouldn't force the usage of unnecessary / unfitting features.
const-correctness compliance.	1	Applying `const` not only where it could be, but also where, in principle, the data shouldn't be modified.
Avoiding unnecessary copies.	1	According to information shared during classes. Applies to types bigger than primitives. It is advised to copy `int`s or `double`s instead of passing them as `const&` (of course constant primitives should be passed by `const`).
Doxygen compliant documentation.	4	
Clarity of error messages.	3	
Correct implementation of encoding and decoding the source file.	7	
Correct implementation of timestamp handling.	5	
Correct implementation of command #1.	2	
Correct implementation of command #2.	3	

Correct implementation of command #3.	5	
Correct implementation of command #4.	2	
Correct implementation of command #5.	2	
Correct implementation of command #6.	1	
Correct implementation of command #7.	3	
Clarity of the main menu. Intuitiveness of the program.	6	The program should be easy and intuitive to use and navigate. Its message prompts should be clear and informative.

Table 1.Grading criteria.

7. Defense

Student will be questioned about the finer details of their implementation. Failure to understand and identify any piece of code may be the basis for **failing the entire project**. It is suggested that students who finished the project long before the due date made sure to remember the finer details of their work.

8. Additional remarks

It is forbidden to use solutions that are not well understood. Every tool must be mastered by the student. Using the tools not shown during the class and lectures is allowed but bear in mind that during the defense student might be questioned about those elements too.

In case of any ambiguities please consult the content and requirements of the project with the tutor.