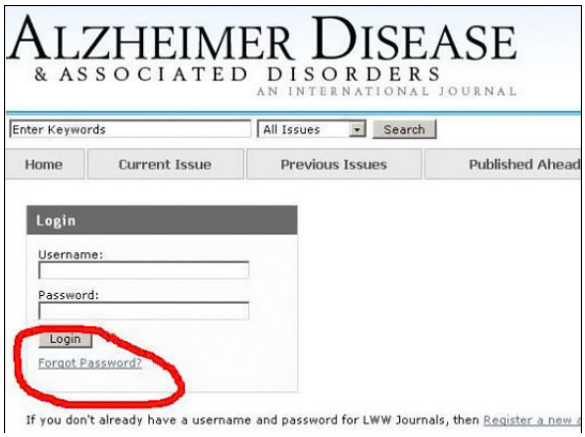
Travail pratique #03



Objectifs

1. Utiliser correctement une fonction de hashage appropriée
2. Comprendre et mettre en application des bonnes techniques de gestion de mot de passe
3. Expérimenter avec des techniques simples pour trouver un mot de passe hashé
4. Mettre en pratique la matière vue en classe

Règles importantes

1. Chaque classe doit avoir son propre fichier *.h* et *.cpp*
2. N'oubliez pas les *"include guard"*
3. Prenez grand soin de respecter la casse pour les noms de classe, méthode, etc
4. Veuillez commenter votre code *intelligemment*
5. Votre code doit être de qualité exemplaire
6. **Votre code doit compiler sans erreurs pour être corrigé**

Instructions

Voici un bout de code qui vous est fourni :

Listing 1 – Code de base

```
#include <string>
#include <map>

class PasswordManager
{
    public:
        // Charge en mémoire le contenu du fichier, on assume en tout temps que
        // le fichier existe et qu'il contient des données valides
        PasswordManager(const std::string& fichier);

        // Lors de la destruction, effectuer une sauvegarde automatique de la liste
        // des usagers/mdp dans le fichier utilisé dans le constructeur
        ~PasswordManager();

        // Retourne le nombre d'usagers
        int Count() const;

        // Retourne vrai si l'usager existe, faux sinon
        bool Existe(const std::string& nom) const;

        // Sauvegarder la liste des usagers/mdp dans le même fichier
        // utilisé dans le constructeur
        void Sauvegarder() const;
```

```
        // Ajouter un usager, retourne faux si l'usager existe déjà, vrai sinon
        bool AjouterUsager(const std::string& nom, const std::string& mdp);

        // Enlève un usager, retourne faux si l'usager est inexistant, vrai sinon
        bool EnleverUsager(const std::string& nom);

        // Cette méthode doit tenter de trouver les mots de passe faibles parmis la
        // liste des usagers et afficher les usagers et mots de passe trouvés (en clair)
        // dans la console.
        // Pour cette fonction, un mot de passe est considéré comme faible si vous êtes
        // capable de le trouver par brute force (5 caracteres de long max excluant les salt,
        // et seulement les caractères a-z0-9) OU que le mot de passe fasse partie
        // du dictionnaire "dict.txt". Un exemple de dictionnaire est fourni avec ce travail,
        // mais prenez pour acquis que je pourrais tester avec un autre fichier portant le
        // même nom.
        void TrouverMotDePasseFaibles() const;

        // Retourne vrai si la combinaison usager/mdp est bonne, faux sinon
        bool Authentifier(const std::string& usager, const std::string& mdp) const;

        // Change le mot de passe d'un usager. Retourne faux si l'usager n'existe pas,
        // vrai sinon
        bool ChangerMotDePasse(const std::string& usager, const std::string& mdp);

    private:
        std::map<std::string, std::string> m_listeUsagers;
};
```

Vous devez implémenter le code des méthodes de la classe `PasswordManager` pour être en mesure de gérer une liste d'usager et de mot de passe. Le stockage des informations se fait dans un simple fichier texte, chaque ligne contenant les informations d'un utilisateur au format suivant :

Listing 2 – Format du fichier des usager

```
usager1:xxxxxxxxxxxxxxxxxxxxxxxxxx
usager2:yyyyyyyyyyyyyyyyyyyyyyyyyy
usager3:zzzzzzzzzzzzzzzzzzzzzzzzzz
```

Le nom d'usager et le mot de passe sont séparés par un `':'` sans espaces. Le format du mot de passe dépendra de votre implémentation.

Outre les consignes déjà fournies en commentaire dans le code ci-dessus, voici des remarques importantes :

1. **Il est fortement suggéré de consulter les liens données en référence vers la fin de ce document avant de commencer le travail.**
2. Vous ne pouvez pas changer la signature des méthodes publiques listées ci-haut.
3. Aucuns mot de passe ne doit être stocké en clair dans le PasswordManager ni sur disque.
4. Vous devez utiliser les techniques vues en classe, notament l'utilisation d'un salt unique par usager, ainsi que l'ajout d'un salt global (pepper) et identique pour tout les usagers et ne se retrouvant pas dans le fichier.
5. Vous devez choisir un algorithme de hashage qui selon vous serait un bon choix pour le hashage des mots de passe. Plusieurs choix sont bons, vous devez par contre être capable de le justifier.
6. Pour le hashage du mot de passe, vous pouvez utiliser un bout de code provenant d'internet correspondant à l'algorithme que vous aurez choisi. Vous devez par contre vous assurer de respecter la license du code en question, ET de mettre en commentaire dans le code l'adresse internet du code original.
7. Vous devez faire des jeux de test pour valider que votre code fonctionne correctement. Ces jeux de test doivent être remis avec le reste de votre projet.
8. **Veuillez justifier à l'aide d'un texte** en commentaire dans le haut de votre fichier `main.cpp` les choix d'algorithmes que vous avez fait (fonction de hash, taille du salt, etc) et en quoi cela rend votre gestion de mot de passe sécuritaire. Pourquoi est t'il le mieux approprié selon vous pour l'usage que vous en faites.

Références

1. Hash function security summary - `https://en.wikipedia.org/wiki/Hash_function_security_summary`
2. List of hash functions - `https://en.wikipedia.org/wiki/List_of_hash_functions`

3. Comparison of cryptographic hash functions - `https://en.wikipedia.org/wiki/Comparison_of_cryptographic_hash_functions`

Remise

Vous devez me remettre sur vortex votre solution *Visual studio* complète **une fois la solution nettoyée** :

1. Menu Générer, Nettoyer la solution
2. Effacer manuellement les autres fichiers inutiles