Compte rendu
TP C++

Andres Baptiste & Imhoff Guillaume

Sommaire :

1. Introduction

Dans le cadre du module d'informatique et dans la logique de notre apprentissage en programmation orienté objet, nous sommes amenés à réaliser deux TPs parmi six autres triés deux à deux par niveau de difficulté. Après une courte concertation nous avons choisi de réaliser le TP 4 de niveau intermédiaire et le TP 5 de niveau intermédiaire à élevé qui ont tous deux pour objectif d'implémenter plusieurs algorithmes de chiffrement/déchiffrement de données dans un programme orienté objet. Nous avons choisi ces TP à la fois par intérêt pour le chiffrement et parce que nous avons estimé qu'ils correspondaient à notre niveau.

Après avoir explicité notre méthode de travail, nous verrons dans un premier temps, la création de la classe basique "Encrypt" qui nous permettra de réaliser nos futures classes avec différents algorithmes de chiffrement/déchiffrement de données. Dans un second temps, nous verrons les deux algorithmes de chiffrement, Caesar et Vigenère ainsi que la création de leur classe. Enfin, nous étudierons la célèbre machine Enigma utilisée par les allemands lors de la 2ème guerre mondiale.

En parallèle, nous pourrons trouver l'ensemble des classes citées dans le compte rendu dans le GitHub : https://github.com/LeBourguignon/TP-3A dans leur répertoire de TP respectif puis dans leur répertoire "ProjetMakeFile". Nous retrouverons aussi des exemples d'utilisation dans leur .h respectifs qu'il suffira de mettre dans le main.cpp. Enfin nous assurons une bonne compilation et un bon fonctionnement des différentes classes.

2. Méthode de travail

Nous avons immédiatement remarqué qu'il serait intéressant de réfléchir conjointement à l'implémentation de certaines classes, et de nous en répartir certaines. Après avoir mis en place le dépôt GitHub, nous avons ainsi réalisé les classes BasicEncypt et Encrypt ensemble, sur le même poste, avant de nous partager les classes suivantes. Enfin, nous avons de la même façon réalisé conjointement la classe Enigma en cause de sa difficulté.

Par habitude, nous avons choisi d'utiliser l'IDE, Visual Studio 2019, et de créer pour chacun des TPs, un projet Visual Studio et un projet MakeFile sur le dépôt permettant de compiler le code de façon différente.

3. La classe basique "Encrypt"

3.1. La classe BasicEncrypt

En s'appuyant sur les informations présentées dans l'énoncé, nous avons tout d'abord créé la classe BasicEncrypt, possédant donc 2 variables protégées _plain et _cypher, ainsi qu'un ensemble de méthodes publiques : les getters (plain(), cypher()), les setters (updatePlain(), updateCypher()) ainsi que encode() et decode() permettant

respectivement d'encoder le plain dans le cypher, ou à l'inverse de décoder le cypher dans le plain (en l'occurrence sans modification puisqu'on ne met en place ici que l'architecture de nos futures classes de cryptage).

Sont aussi rédigées 2 fonctions read() et write() permettant de sauvegarder et de lire des chaînes de caractères dans des fichiers (ces deux fonctions sont toujours disponibles dans le code rendu mais ont été mises en commentaires après leur réécriture dans les fichiers encrypt.h et encrypt.cpp). Le tout fonctionne parfaitement.

3.2. La classe Encrypt

La classe Encrypt est une réécriture quasiment à l'identique de BasicEncrypt en définissant toutefois encode() et decode() comme virtuelles. Nous avons envisagé de rendre ces deux méthodes virtuelles pures, afin d'obliger leur redéfinition lors de l'héritage, mais avons finalement convenu que les définir comme simplement virtuelles offrait une plus grande flexibilité. Le tout fonctionne parfaitement.

4. Deux algorithmes de chiffrement

4.1. La classe Caesar

Nous nous attaquons maintenant au chiffrement et déchiffrement des données avec l'algorithme de Caesar. Nous avons commencé par créer une classe héritant de Encrypt et redéfinit les méthodes encode() et decode() ce que nous ferons aussi pour les classes suivantes.

Le chiffrement de César consiste à effectuer un décalage de n positions vers la droite de toute lettre du message à encoder. Nous considérerons dans un premier temps que les messages sont constitués uniquement de lettres minuscules, tous les autres caractères étant ignorés.

Nous avons donc défini les méthodes de chiffrement et de déchiffrement. Les méthodes vont tout d'abord prendre les caractère un part un, vérifier s'ils sont bien des lettres minuscules enfin réaliser un décalage de n positions dans l'alphabet à droite pour encode() (aprés z nous retournons à a) et à gauche pour decode(). Le tout fonctionne parfaitement.

4.2. La classe Caesar2

Le but de cette classe est de reprendre le chiffrement de Caesar vu précédemment mais pour l'ensemble du code ascii. Il nous suffit donc de retirer la vérification de lettre minuscule et de réaliser le décalage dans les caractères ascii. Le tout fonctionne parfaitement.

4.3. La classe Vigenere

Le chiffrement de Vigenère suit le même principe de base que le chiffrement de César (on décale chaque lettre du texte à encoder d'un certain nombre de lettres dans l'alphabet) sauf que le décalage de chaque lettre du texte dépend maintenant de sa position dans le texte.

Pour implémenter en C++ ce chiffrement, nous avons évidemment créé une classe héritée de la classe Encrypt, en écrivant dans la fonction encode() une version optimisée pour le C++ du code Python présenté dans le document cryptographie.pdf fourni avec l'énoncé. La fonction decode() à quant à elle été rédigée par nos soins, en appliquant les étapes de l'algorithme précédent dans l'ordre inverse. Le tout fonctionne parfaitement.

5. La classe Enigma

Cette classe a pour objectif d'implémenter un chiffrement semblable à celui utilisé par la machine Enigma, tel que décrit par le document annexe à l'énoncé.

Comme pour les classes précédentes, nous commençons par créer une classe héritière de Encrypt. Dans un premier temps nous redéfinissons les méthodes encode() et decode() en simulant l'action d'un seul rotor. Nous réalisons cela en prenant la position dans l'alphabet de la lettre à chiffrer puis on décale du nombre de lettres chiffrées avant et on prend la lettre à cette position dans la clé donnée lors de l'initialisation (par défaut la clé est l'alphabet).

Nous voulons ensuite simuler l'action de plusieurs rotors. Nous avons donc créé une fonction simulant un rotor dans les méthodes protégées rotorEncode() et rotorDecode() avec l'algorithme décrit précédemment. En parallèle nous avons ajouté la possibilité d'entrer jusqu'à 5 clés lors de l'initialisation qui induit 5 rotors et modifié les méthodes encode() et decode() pour utiliser les différents rotors en prenant en compte les différentes vitesse de rotation des rotors (un cran par caractère pour le premier rotor, un cran tous les 26 caractères pour le second, et caetera). Le tout fonctionne parfaitement.

6. Conclusion

Les deux TP nous sont parus très intéressants, tant du côté de l'introduction au chiffrement que de celui de l'application de la programmation orientée objet, avec l'usage notable d'héritage et des chaînes de caractère les "std::string". Nous sommes satisfaits d'être parvenus à réaliser ces 2 TP sans rencontrer de blocage insurmontable et en obtenant les résultats escomptés.