ULB UNIVERSITÉ LIBRE DE BRUXELLES

Projet langage de programmation

Secundar Ismael

504107

16 Mai 2021

Table des matières

1 Introduction

L'objectif du projet du cours INFO-F105 est d'implémenter en C++ et en Python le chiffrement de message entré par l'utilisateur.

Ce rapport a pour but de comparer les différentes améliorations implémentées. Pour cela, le temps d'exécution avant l'amélioration et après l'amélioration seront pris en compte.

Avant cela, nous allons d'abord résumer brièvement en quoi ce projet consistait lors des différentes parties.

Lors de la partie 1, nous avons uniquement implémenté 3 fonctions. La première était Sanitize. Cette fonction avait pour but de nettoyer un message encodé par l'utilisateur en remplaçant les minuscules par des majuscules et en enlevant les chiffres. La seconde fonction était Caesar. L'utilisateur avait le choix de choisir le décalage et le nombre de décalage qu'il voulait pour son message. La dernière fonction implémentée lors de cette partie était Send. Elle permettait d'écrire le message dans un fichier choisi par l'utilisateur.

Pour la partie 2, nous avons ajouté la fonction Enigma. Cette fonction permettait à l'utilisateur de chiffrer son message. C'est lors de cette partie que nous avons eu beaucoup de mal parce qu'il fallait gérer un header avec des char-array.

Pour la dernière partie du projet, nous avons changé la structure de notre code en classe, mais avant cela il nous fallait mesurer l'efficacité de notre code sans classe. C'est essentiellement sur cela que notre rapport sera basé, nous allons comparer l'efficacité de notre code avant qu'il ne soit mis en classe et après qu'il soit mis globalement en classe.

Pour terminer l'introduction, nous avons mis ci-dessous les commandes afin d'exécuter le code mis sur l'UV.

1. Pour C++, Une fois sur le terminal:
   - g++ -shared -fpic enigma.cpp enigma.hpp -o enigma.so
   - g++ enigma.cpp send.cpp caesar.cpp sanitize.cpp send_auxiliary.cpp -o send -Wall -Wextra
2. Pour César:
   - ./send D 3 /tmp/f105
3. Pour Enigma:
   - ./send enigma /tmp/f105
4. Pour la partie sur Python, pour décrypter César
   - python recv.py D 3 /tmp/f105
5. Pour la partie sur Python, pour décrypter Enigma
   - python recv.py enigma /tmp/f105

2 Méthodes

Afin d'améliorer la qualité du code, nous avons transformé le code en classe. Nous avons créé un fichier en C++ pour générer des mots aléatoires d'une longueur donnée.

2.1 Changement en classe

Pour le changement en classe, nous avons divisé notre code en plusieurs parties. De base, nous avions uniquement un seul header, enigma.hpp. Nous avons maintenant send_auxiliary.hpp, pour alléger le fichier

send.cpp. Ensuite, nous avons mis la fonction Sanitize dans un header aussi. Pour enigma, nous avons changé les struct en classe. De ce fait, il n'y a plus aucun struct mais bien une classe Enigma. En ce qui concerne la partie sur python, recv.py contient maintenant une classe nommée Recv.

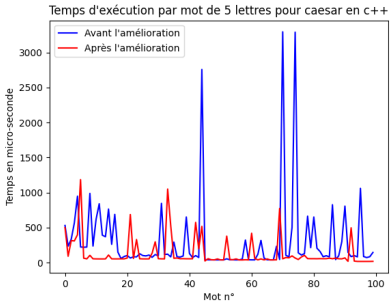
2.2 Autres améliorations

Dans l'ensemble du coup les boucles for ont été utilisés dans la mesure du possible pour éviter les répétitions. L'utilisation des variables magiques a été limité. Un try, catch a été utilisé pour éviter d'avoir un While True à l'infini. Il y avait des warning lors de la partie 2, maintenant, -Wall -Wextra ne donnent pas de Warning.

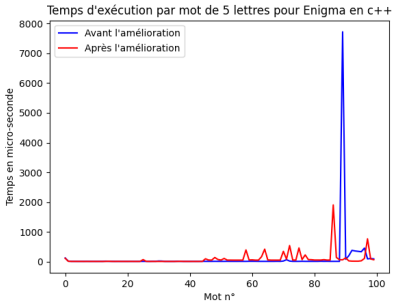
2.3 Temps d'exécution

Pour mesurer le temps d'exécution, nous avons divisé cette tache en deux. La partie python et la partie C++. Nous avons généré des mots aléatoires d'une longueur de 5 lettres et de 10 lettres sans chiffres et sans espaces. Les tests ont été effectués sur 100 mots. En premier lieu, 100 mots de 5 lettres ensuite, 100 mots de 10 lettres. Sur les graphiques cela a bien été détaillé.

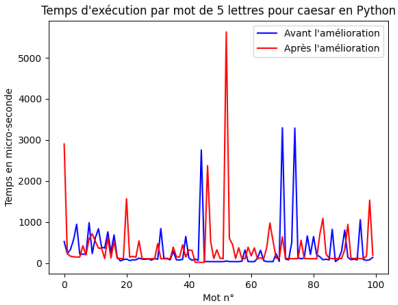
3 Résultats



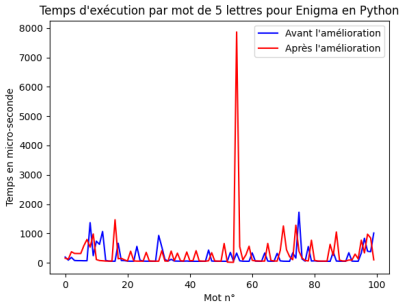
(1) Caesar C++



(2) Enigma C++



(3) Caesar python



(4) Enigma python

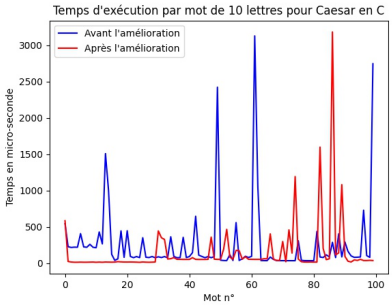
Figure I: Comparaison entre les améliorations avant et après pour des mots d'une taille de 5 caractères

Tableau de comparaison, mots de 5 lettres		
	Avant	Après
Enigma python	μ : 202 | σ : 296	μ :339 | σ : 817
Caesar python	μ : 372 | σ : 738	μ :378 | σ : 684
Enigma C++	μ : 118 | σ : 768	μ :91 | σ : 217
Caesar C++	μ : 308 | σ : 555	μ :132 | σ : 208

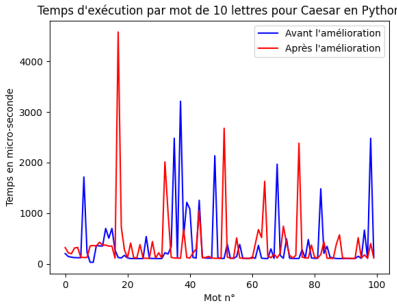
Table 1: Moyenne μ et écart-type σ pour 100 mots de 5 lettres

Tableau de comparaison pour des mots d'une taille de 10 lettres		
	Avant	Après
Enigma python	μ : 359 | σ : 571	μ :365 | σ : 604
Caesar python	μ : 359 | σ : 571	μ :365 | σ : 604
Enigma C++	μ : 158 | σ : 332	μ :154 | σ : 193
Caesar C++	μ : 262 | σ : 499	μ :150 | σ : 386

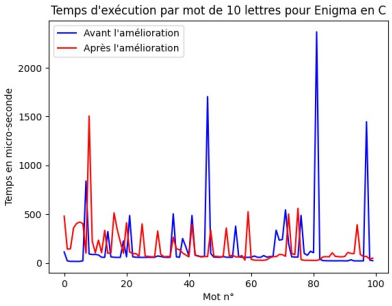
Table 2: Moyenne μ et écart-type σ pour 100 mots de 10 lettres



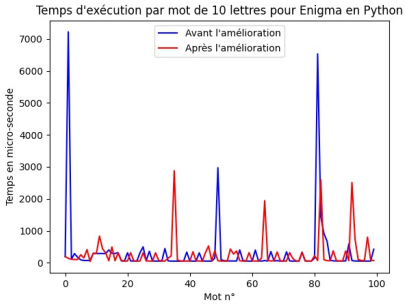
(1) Caesar C++



(2) Caesar python



(3) Enigma C++



(4) Enigma python

Figure II: Comparaison entre les améliorations avant et après pour des mots d'une taille de 10 caractères

4 Discussion

4.0.1 Performance

Nous observons sur la figure I que le temps d'exécution en C++ et en python se sont globalement amélioré. On pourrait conclure que la transformation en classe a été efficace en C++ et sur Python mais d'après la Table 1, l'on peut voir que l'amélioration en classe a été bénéfique en C++ mais pas sur Python. Peut-être qu'en python il aurait fallut laisser l'ancienne version sans classe pour gagner en optimisation et en performance. D'ailleurs, la Table 2 vient confirmer cela. Une fois que nous agrandissons la taille des mots, les résultats deviennent plus clair. L'on peut voir effectivement que l'amélioration a été fructueuse en C++ uniquement.

D'après les graphiques que nous avons obtenus. Il arrivait parfois d'avoir un temps hors du commun pour un mot. Par exemple, sur le graphique 2 de la Figure I, le temps d'exécution pour le mot n°89, le temps d'exécution était de 7717 micro-seconde. Nous avons été voir quel mot cela était, il s'agissait du mot 'eEHBr'. Nous pensions que cela était peut-être une anomalie au niveau des tests mais finalement tout semblait être correcte. Le mystère de cette hausse restera inconnu tout comme la hausse sur le graphique 4 de cette même figure.

4.0.2 Difficultés

La librairie Random ou Names de Python ne sont pas sur C++. Cela nous a pris pas mal de temps avant de comprendre comment générer des mots aléatoires sur C++. Par ailleurs le fait de gérer plusieurs fichiers était un calvaire. Pycharm est beaucoup plus accessible que Virtual Studio code comme IDE. Je trouve que Virtual Studio code est bien mais il est peut-être plus maniable pour les habitués. Les erreurs en C++ ne sont pas souvent facile à résoudre. Le fait de devoir compiler avant d'exécuter les programmes à chaque fois prenait un temps fulgurant vue que l'on devait utiliser la machine virtuelle. Le système d'exploitation Mac IOS ne permettait pas de compiler plusieurs fichier en même temps. D'où le fait qu'il a fallut utiliser explicitement la machine virtuelle.

5 Conclusion

Pour conclure, nous pouvons dire que les résultats nous amènent à dire que les classes en C++ sont bien plus performantes qu'en Python. Le fait d'avoir découpé la partie du code en C++ en plusieurs fichiers et headers a permis d'alléger le temps d'exécution du programme.