INF3135 – Construction et maintenance de logiciels
TP2 - Automne 2021

Décryption du chiffrement de César

Vous devez concevoir un logiciel pour décrypter des fichiers contenant des messages codés selon le Chiffrement de César.

Votre logiciel sera lancé à la console en recevant en paramètre un chemin vers un fichier d'entrée et un chemin vers un fichier de sortie. Le fichier d'entrée contiendra le message encrypté. Le fichier de sortie devra contenir le message décrypté.

Exemple d'exécution du logiciel :
```
bash> ./cesar entree.txt sortie.txt
```

Un fichier d'entrée vous sera fourni pour vos tests. Aucune ligne de dépassera 80 caractères.

Le Chiffrement de César est une méthode d'encryption par décalage de lettre : on remplace une lettre par une autre lettre en fonction d'un décalage prédéterminé. Ainsi : A devient D, B devient E, C devient F, Z devient C et ainsi de suite.

Uniquement les lettres doivent être décryptées. Tous les autres caractères doivent être produits tel quel dans le fichier de sortie, sans transformation. La casse doit être également préservée.

Exigences du code source

Vous devez appliquer les exigences suivantes à votre code source.

- L'indentation doit être de 3 espaces. Aucune tabulation n'est permise dans l'indentation.
- Votre code doit être découpé en fonctions d'une longueur ne dépassant pas 10 lignes par fonction.
- Vous devez adapter une approche de programmation modulaire. Utilisez de fichier d'en-tête (.h) pour représenter vos interfaces et cacher vos implémentations dans les fichiers (.c). Vos modules devraient suivre le standard vu en classe.
- Les erreurs systèmes doivent être gérées correctement (ouverture-fermeture de fichier).
- Les identifiants de fonctions et variables doivent être en ***snake_case***.
- Vous devez rédiger suffisamment de « bons » tests unitaires afin d'acquérir une couverture de tests satisfaisante sur tout votre projet.
- Vous devrez utiliser **`bats`** comme cadre de test de votre logiciel
- Une attention particulière doit être apportée à la lisibilité du code source.
- Vous devez ajouter une intégration continue (*`GitLab-CI`*) de votre projet.

Exigences techniques (Pénalité 20%)

- Votre travail doit être rédigé en langage C et doit compiler sans erreur et sans avertissement (compilation avec l'option -Wall) sur le serveur Java de l'UQAM (java.labunix.uqam.ca). Pour vous y connecter, vous devez connaître votre CodeMS (de la forme ab123456) ainsi que votre mot de passe (de la forme ABC12345)

- Votre dépôt doit se nommer **exactement** inf3135-automne2021-tp2

- L'URL de votre dépôt doit être **exactement** https://gitlab.info.uqam.ca/<utilisateur>/inf3135-automne2021-tp2 où <utilisateur> doit être remplacé par votre identifiant

- Votre dépôt doit être **privé**

- Les usagers *@correcteurs* et *dogny_g* doivent avoir accès à votre projet comme *Developer*

Remise

Le travail est automatiquement remis à la date de remise prévue. Vous n'avez rien de plus à faire. Assurez-vous d'avoir votre travail disponible sur votre branche master qui sera considérée pour la correction. Tous les *commit*s **après le 14 novembre 2021 à 23:59** ne seront pas considérés pour la correction.

Barème

Critère	Points
Fonctionnalité	/50
Qualité du code	/10
Utilisation de git	/5
Tests	/15
GitLab-CI	/5
Makefile	/10
Documentation	/5
Total	/100

Plus précisément, les éléments suivants seront pris en compte:

- Fonctionnalité (50 points): Tous les programmes compilent et affichent les résultats attendus.

- Qualité du code (10 points): Les identifiants utilisés sont significatifs et ont une syntaxe uniforme, le code est bien indenté, il y a de l'aération autour des opérateurs et des parenthèses, le programme est simple et lisible. Pas de bout de code en commentaire ou de commentaires inutiles. Pas de valeur magique. Le code doit être bien factorisé (pas de redondance) et les erreurs bien gérées. La présentation doit être soignée.

- Documentation (5 points): Le fichier README.md est complet et respecte le format Markdown. Il explique comment compiler et exécuter vos programmes ainsi que toutes les autres cibles demandées.

- Utilisation de Git (5 points): Les modifications sont réparties en *commits* atomiques. Le fichier .gitignore est complet. Utilisation des branches. Les messages de *commit* sont significatifs et uniformes.

- Tests (15 points): Le code de test est propre, fournissant une couverture suffisante au projet.

- Makefile (10 points): Le Makefile supporte les cibles `all`, `start`, `build`, `link` et `clean`. L'appel à `make` doit compiler et construire l'exécutable.

- GitLab-CI (5 points): Votre projet doit supporter le système d'intégration continue de GitLab (GitLab-CI) qui construit et roule tous vos tests à chaque commit sur la branche master.