INF3135 – Construction et maintenance de logiciels
TP2 - Automne 2022

Résolution de mots cachés

Vous devez concevoir un logiciel pour découvrir un mot caché dans une grille.

Votre logiciel sera lancé à la console en recevant en paramètre un nom de fichier. Le fichier contiendra la grille de lettres et les mots à trouver. Une fois les mots trouvés, le logiciel devra afficher à l'écran la solution : le mot trouvé à partir des lettres non utilisées dans la grille.

La grille fait une taille de 12 lettres par 12 lettres.

Exemple d'exécution du logiciel :
bash> ./motcache entree.txt
MANDOLINE
bash>

Un fichier d'entrée vous sera fourni pour vos tests. Les 12 premières lignes du fichier contiendront les 12 lignes de la grille. Ensuite, le fichier contiendra une ligne vide et finalement la liste de mots à trouver. Chaque mot sera sur une ligne différente.

Les lettres d'un mot peuvent apparaître à l'envers dans la grille. En réalité, quatre directions sont possibles :

- de gauche à droite;
- de droite à gauche;
- de haut en bas;
- de bas en haut.

Il n'est pas nécessaire de faire les diagonales, ce cas ne sera pas testé.
Une lettre de la grille peut être utilisée par plus d'un mot. Lorsque tous les mots ont été trouvés, la réponse au mot caché est trouvée en affichant à l'écran toutes les lettres non utilisées, en parcourant la grille de gauche à droite, de haut en bas.

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 acceptable. Pas de fonctions de plus de 10 lignes.
- Chaque fonction doit être documentée avec un commentaire expliquant l'objectif de la fonction, les paramètres et la valeur de retour (si applicable). Le nom de la fonction doit être significatif.
- N'utilisez pas de variables globales (sauf errno).
- Les erreurs systèmes doivent être gérées correctement.
- 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 identifiants de fonctions et variables doivent être en snake_case.
- Une attention particulière doit être apportée à la lisibilité du code source.
- Vous devrez utiliser **`bats`** comme cadre de test pour tester votre logiciel
- Vous devez ajouter une intégration continue (GitLab-CI) dans votre projet.
- Vous devez fournir un Makefile qui exprime les dépendances de façon complète.

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-aut2022-tp2`
- L'URL de votre dépôt doit être **exactement** `https://gitlab.info.uqam.ca/<utilisateur>/inf3135-aut2022-tp2` où `<utilisateur>` doit être remplacé par votre identifiant
- Votre dépôt doit être **privé**

Les usagers *`@zavaleta_chuquicaja.willy`*, *`@bernardos.pedro_luis`* 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 ou main`)* qui sera considérée pour la correction. Tous les *commit*s **après le 13 novembre 2022 à 23:55** ne seront pas considérés pour la correction.

Barème

Critère	Points
Fonctionnalité	/40
Qualité du code	/15
Utilisation de git	/5
Tests	/15
GitLab-CI	/10
Makefile	/10
Documentation	/5
Total	/100

Plus précisément, les éléments suivants seront pris en compte:

- **Fonctionnalité (40 points)**: Tous les programmes compilent et affichent les résultats attendus.
- **Qualité du code (15 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. Les fonctions sont bien documentées.
- **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, les cas de tests sont pertinents et couvrent l'ensemble des fonctionnalités.
- **Makefile (10 points):** Le Makefile doit supporter au moins les cibles *`compile, link et test.`* L'appel à `make` doit compiler et construire l'exécutable.
- **GitLab-CI (10 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.