INF3135 – Construction et maintenance de logiciels
TP3 - Été 2023

Recherche

Vous devez concevoir un logiciel servant à effectuer des recherches dans une banque de recettes.

Votre logiciel sera lancé à la console en recevant en paramètre un nom de fichier. Le fichier contiendra la banque de recettes dans laquelle nous effectuerons les recherches. Chaque ligne du fichier contiendra une recette avec les catégories dans lesquelles elle doit apparaître. Une recette est obligatoirement dans une catégorie mais pourrait aussi apparaître dans plusieurs catégories. Les catégories apparaissent entre crochets, après le nom de la recette. Vous pouvez assumer que la banque de données sera bien formée et sans doublons. Les lignes du fichier ne dépasseront pas 120 caractères.

Exemple de banque de données :

```
Poulet au romarin [poulet] [bbq]
Boeuf au satay [boeuf] [asiatique]
Salade du jardin [vegetarien]
Poulet crapaudine [poulet]
Pho [soupe] [asiatique]
Authentique gibelotte des iles de Sorel [soupe] [poisson] [terroir]
```

Au démarrage de l'application, la banque de données doit être chargée en mémoire dans une structure de données. Ensuite, l'application doit saisir au clavier des instructions de recherche et afficher les résultats. L'affichage des résultats ne consiste qu'à afficher les recettes correspondant au critère de recherche, en ordre alphabétique.

L'application devra supporter 2 types de recherche : par catégorie et par catégorie avec un mot-clé. La recherche est toujours insensible à la casse.

Une recherche par catégorie ne prendra que le nom de la catégorie et affichera les recettes dans cette catégorie. Si la catégorie n'existe pas, un message doit être affiché.

Une recherche par catégorie et mot-clé doit prendre le nom de la catégorie et un mot-clé, séparés par un ou plusieurs espaces. Le mot-clé est un mot qui doit être présent dans le nom de la recette. Si la catégorie n'existe pas, un message doit être affiché. Si aucune recette ne correspond aux critères, un message doit être affiché.

Toute autre combinaison sera considérée invalide et sera rejeté par l'application.

Exemple d'exécution du logiciel :

```
Entrez votre critère de recherche : Salami
Catégorie inexistante.
Entrez votre critère de recherche : POULET
Poulet au romarin
Poulet crapaudine.
Entrez votre critère de recherche : poulet crapaud
Poulet crapaudine.
Entrez votre critère de recherche : poulet au crapaud
Recherche invalide.
Entrez votre critère de recherche : poulet salade
Aucun résultat trouvé.
Entrez votre critère de recherche : salade verte
Catégorie inexistante.
```

Statistiques

Lorsque le logiciel sera invoqué avec l'option -S, il accumulera des statistiques sur les données du fichier d'entrée et écrira ces statistiques dans un fichier de sortie spécifié à la console.

Exemple d'exécution du logiciel avec l'option -S :

```
bash> ./recherche liste.txt –S stats.txt
```

Si le fichier de sortie n'existe pas, il sera créé; s'il existe, il sera écrasé.

Voici les statistiques à produire :

- Le nombre de lignes dans le fichier d'entrée
- Le nombre de mots sans doublons
- Le nombre de mots avec doublons
- La lettre la plus fréquente (sans considérer les doublons)
- Le nombre de catégories
- Le nombre de recettes
- La catégorie qui a le plus grand nombre de recettes
- La recette la plus longue (en termes de nombre de caractères)

Exigences de conception

La structure de données principale doit être une liste chaînée contenant les catégories. La liste de catégories doit toujours être triée en ordre alphabétique.

Chaque catégorie doit également contenir une liste chaînée de recettes. Les listes de recettes doivent également être triées en ordre alphabétique.

Les structures de données utilisées pour accumuler les catégories et les recettes doivent être allouées dynamiquement.

Le logiciel doit être découpé en modules. On devrait au minimum, retrouver un module pour chaque élément suivant :

- Le main
- La liste chainée
- La gestion des statistiques
- Les tests

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 paramètres reçus par le main (argc et argv) doivent toujours être validés.
- Vous devez utiliser les branches pour la gestion des sources. En plus de la branche principale, vous devrez avoir une branche par module, qui seront fusionnées à votre branche principale à travers une demande d'unification (*`merge request`*)
- Votre système ne doit avoir aucune fuite de mémoire.
- Vous devez rédiger suffisamment de « bons » tests unitaires afin d'acquérir une couverture de tests satisfaisante sur tout votre projet.
- Vous devrez utiliser **`CUnit`** comme cadre de test de votre logiciel. Une couverture de test minimale de 70% est requise sur votre logiciel.
- 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)

- Le travail peut être réalisé en équipe d'au plus 3 personnes. Attention, même si vous travaillez en équipe, vous pouvez avoir des notes différentes (je vérifie qui fait quoi).

- Votre dépôt doit se nommer **exactement** inf3135-ete2023-tp3

- L'URL de votre dépôt doit être **exactement** https://gitlab.info.uqam.ca/<utilisateur>/inf3135-ete2023-tp3 où <utilisateur> doit être remplacé par l'identifiant du chef de groupe

- Votre dépôt doit être **privé**

- Les usagers *@lapointe.gabriel.2* 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 13 août 2023 à 23:59** ne seront pas considérés pour la correction.

Barème

Critère	Points
Fonctionnalité	/40
Utilisation de git	/5
Tests	/20
Exigences de conception	/15
Makefile, Qualité du code, Documentation, GitLab-CI	/20
Total	/100

Plus précisément, les éléments suivants seront pris en compte:

- Fonctionnalité (40 points): Tous les programmes compilent et affichent les résultats attendus.
- Exigences de conception (15 points): Le respect des exigences de conception et des structures de données exigées. La modularité de la conception

- Utilisation de Git (5 points): Les modifications sont réparties en *commits* atomiques. Le fichier .gitignore est complet. Utilisation des branches et des requêtes d'intégration (merge request). Les messages de *commit* sont significatifs et uniformes.
- Tests (20 points): Bonne utilisation du cadre de test Cunit, Le code de test est propre avec une couverture de tests de 70 % sur le projet.
- Makefile, Documentation, GitLab-CI, Qualité du code (20 points) :
    - Documentation : Le fichier README.md est complet et respecte le format Markdown. Il explique comment compiler et exécuter vos programmes. Ce fichier doit avoir une section nommée « équipe », qui liste les membres de l'équipe (nom et prénoms, code permanant, courriel) et doit clairement identifier le chef de l'équipe.
    - Makefile : Le Makefile qui met en œuvre de façon adéquate, la construction de votre logiciel et qui génère la documentation en html.
    - GitLab-CI : 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 principale.
    - Qualité du code : Lisibilité, uniformité, testabilité