

Programmation avancée - Projet

Capteurs de pollution

IMA2A3
Juin 2020

Kevin Doolaeghe
God-Belange Aradukunda

Sommaire :

1. Présentation du sujet.....	3
2. Etude du projet.....	3
3. Organisation	4
4. Mise en œuvre	5
a. Interface Homme-Machine.....	5
b. Lecture des fichiers CSV.....	6
c. Boucle principale	6
d. Analyse des requêtes	7
5. Conclusion	7
6. Annexes	8

1. Présentation du sujet

Le but du projet proposé est de réaliser un programme sous forme de lignes de commandes qui permet d'analyser les données récoltées par 449 capteurs déployés dans la ville d'Århus au Danemark.

Le programme doit proposer les fonctionnalités suivantes :

- Le programme peut fonctionner sous deux modes de fonctionnement. On compte donc le mode *batch* qui lit un fichier de requêtes à utiliser et le mode *interactif* qui traite les requêtes spécifiées par l'utilisateur
- Les requêtes doivent respecter la syntaxe suivante
metric [opt] type [longitude latitude radius]
- Le programme doit pouvoir lire les fichiers CSV contenant les données
- Chaque requête retourne un résultat de la forme *avg min max*
- Un fichier *Makefile* permettra de compiler automatiquement le programme
- Un fichier *README.md* sera présent pour expliquer comment compiler et utiliser le programme

2. Etude du projet

Après étude du cahier des charges, il nous faut coder en langage C (le programme doit fonctionner sur l'environnement Linux Debian), un programme permettant d'analyser les données de fichiers au format CSV.

Le programme comporte deux modes de fonctionnement. Le premier est un mode batch qui lit un fichier de requêtes et les traite une à une jusqu'à la fin du programme tandis que le second est un mode interactif où les requêtes sont directement spécifiées par l'utilisateur puis traitées.

Le programme sera capable de reconnaître les requêtes pour extraire les bonnes données et fournir un résultat correct à l'utilisateur. Si une requête n'est pas reconnue, une aide s'affiche. Les requêtes sont stockées dans une liste de chaînes de caractères.

Pour le stockage des données, on utilisera des listes chaînées permettant principalement des ajouts et suppressions rapides. La recherche d'un élément se fait par le parcours entier de la liste mais puisque toutes les données doivent être analysées, ce n'est pas contraignant.

Le code entier du programme se trouve dans un dépôt Git sur Gitlab. Il s'agit d'une plateforme analogue à Github permettant en outre une historisation et une gestion de version pour le développement de logiciels.

Dans le dépôt, on trouve principalement les fichiers sources dans le dossier *src*, les fichiers d'entêtes et les bibliothèques dans le dossier *includes*, un fichier Makefile permettant la compilation automatique des fichiers sources, un fichier README.md expliquant les objectifs du projet et comment compiler et utiliser le programme. Il y a également les fichiers *.clang-format*, *.gitignore* qui servent respectivement au formatage du code, et aux exceptions de Git lors de l'envoi des mises à jour. Le dépôt contient enfin le rapport du projet qui est le fichier présent.

Le code est segmenté en plusieurs fichiers ayant chacun, une fonction précise. Chaque fichier source possède un fichier d'entête qui décrit ce que contient le fichier source et initialise certaines

variables. Cela permet ainsi d'améliorer la lisibilité et la modification/mise à jour du code. Voici la liste des fonctionnalités développées par fichier :

- `display.c` : Fonctions utiles pour l'affichage
- `utils.c` : Fonctions utiles pour le programme
- `date.c` : Structures de données pour les dates
- `record.c` : Structures de données pour les enregistrements
- `list_record.c` : Structures de données pour les listes d'enregistrements
- `str.c` : Structures pour la gestion des chaînes de caractères
- `list_str.c` : Structures pour la gestion des tableaux de chaînes de caractères
- `read_dir.c` : Fonctions pour la lecture de dossiers
- `read_file.c` : Fonctions pour la lecture de fichiers
- `request.c` : Analyse et traduction des requêtes et structures pour les requêtes et le résultat
- `loop.c` : Boucle principale, gestion du mode de fonctionnement
- `main.c` : Point d'amorçage du programme

3. Organisation

Après concertation, le travail est réparti comme présenté ci-dessous.

Kevin	God-Belange
<ul style="list-style-type: none">• Création des structures de données• Lecture de dossiers et de fichiers• Boucle principale	<ul style="list-style-type: none">• Analyse des requêtes• Traitement de la requête• Affichage des résultats

4. Mise en œuvre

a. Interface Homme-Machine

L'interface du programme est en ligne de commande et est identique pour les deux modes d'exécution puisqu'à chaque instruction suit un résultat. Pour le mode interactif, il y a des commandes supplémentaires comme *help* ou *display* pour afficher de l'aide ou afficher la liste entière.

```
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Interactive mode :
> global ozone
112.761693 15 215
> monthly 2014-09 particulate_matter
114.265835 15 215
> help
usage: metric [opt] type [longitude latitude radius]
> Bonjour à tous
> quit
Bye !
```

Le mode *batch* suit le même principe que le mode *interactif* à la différence qu'il s'effectue de manière automatique. Il lit en effet, un fichier de commandes dont le nom est fourni en argument et va traiter les commandes unes à unes. Après chaque commande effectuée, le programme attend une validation de l'utilisateur pour continuer le traitement. Pour cela, celui-ci doit appuyer sur la touche entrée. A la fin de l'exécution de toutes les commandes, le programme s'arrête.

```
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading
Batch mode :
> global particulate_matter
112.092948 15 215

> daily 2014-8-3 nitrogen_dioxide
92.270532 15 215

> monthly 2014-8 carbon_monoxide
109.373000 15 215

> daily 2014-8-2 sulfur_dioxide 10.104 56.231 1000
85.621945 15 215

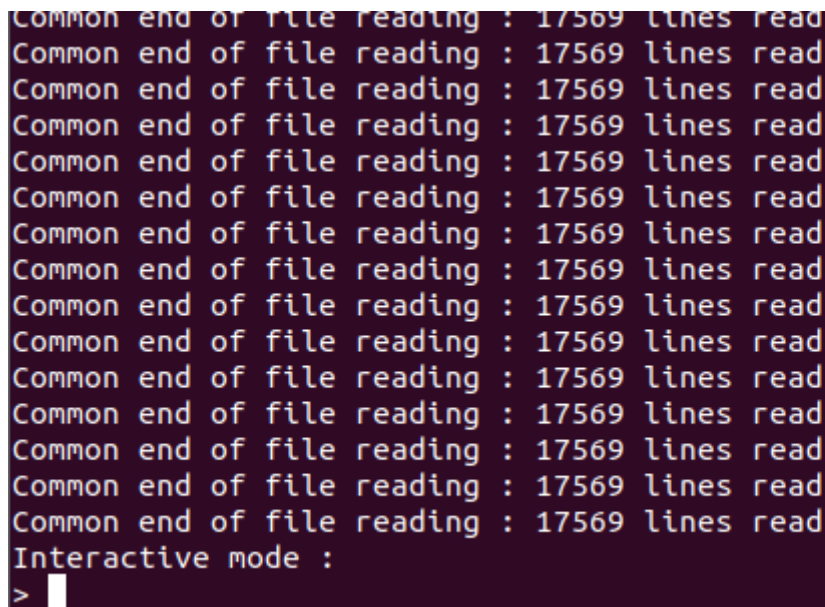
> log 2014-8-1 0:5:0 ozone
61.659243 21 104

Bye !
```

b. Lecture des fichiers CSV

Le programme doit être capable de parcourir un dossier de fichier afin de lire leur contenu. Dans un premier temps, le programme va donc lire les fichiers du dossier en ne gardant que les fichiers concernés puis il va parcourir la liste de fichiers obtenue pour lire les données et les placer dans la liste chaînée.

La lecture d'un fichier se fait ligne par ligne. Chaque ligne est découpée à partir du caractère séparateur (ici, la virgule) ce qui permet d'obtenir les différents champs. Après cela, les données sont regroupées dans une structure de type *Record* représentant un enregistrement pour être ensuite ajoutée à la liste. Le programme imprime à l'écran si un fichier a bien été lu et le nombre de lignes lues.



```
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Common end of file reading : 17569 lines read
Interactive mode :
>
```

c. Boucle principale

La boucle principale commence par faire ce dont les deux modes ont besoin, c'est-à-dire lire les données. Il y a donc initialement la lecture du dossier fourni en argument, puis la lecture des fichiers avec l'ajout des données dans une liste chaînée. Ensuite, par analyse des arguments, on détermine le mode de fonctionnement.

Si le mode choisit est le mode *batch*, alors on effectue la lecture des requêtes du fichier fourni en argument. Le programme entre ensuite dans une boucle qui va traiter chaque requête. Le traitement va afficher la requête, déterminer le résultat (par analyse et recherche) puis afficher celui-ci pour l'utilisateur.

Si d'autre part, le mode choisit est le mode interactif, alors le programme entre directement dans une boucle dans laquelle il attend la requête/commande puis effectue le traitement de celle-ci. L'utilisateur peut entrer les commandes suivantes pour effectuer des opérations dans le programme.

- *help* : permet d'obtenir de l'aide
- *quit* : permet de quitter le programme
- *display* : permet d'afficher la totalité des données en mémoire

d. Analyse des requêtes

Dans la boucle principale, chaque requête saisie ou lue est traitée. Pour cela, la requête est découpée pour qu'après analyse de son contenu, on puisse savoir ce qui est recherché. Dès que la requête est reconnue, on parcourt la liste de données pour construire une liste avec les mesures désirées. Lorsque la liste a été entièrement parcourue, on extrait la valeur moyenne, maximale et minimale de la liste de mesures pour l'afficher à l'écran.

Le découpage des requêtes est plutôt sécurisé car il vérifie que la structure de la requête est correcte. Si ce n'est pas le cas alors, la requête n'est pas traitée. L'algorithme de vérification des requêtes pourrait être davantage amélioré pour rendre le programme plus robuste à des requêtes erronées et ainsi éviter une erreur de segmentation.

5. Conclusion

L'état final du projet respecte tous les points du cahier des charges :

- Le programme peut fonctionner sous deux modes de fonctionnement. On compte donc le mode *batch* qui lit un fichier de requêtes à utiliser et le mode *interactif* qui traite les requêtes spécifiées par l'utilisateur
- Les requêtes doivent respecter la syntaxe suivante
metric [opt] type [longitude latitude radius]
- Le programme doit pouvoir lire les fichiers CSV contenant les données
- Chaque requête retourne un résultat de la forme *avg min max*
- Un fichier *Makefile* permettra de compiler automatiquement le programme
- Un fichier *README.md* sera présent pour expliquer comment compiler et utiliser le programme

Cependant, certaines des fonctionnalités ci-dessus peuvent être améliorées davantage. La reconnaissance des fichiers peut être améliorée et les performances générales du programme pourraient être accrues. La détection des requêtes saisies peut également être améliorée pour éviter le plantage du programme.

Nous avons rencontré quelques soucis de débordements mémoires lors de la conception des listes de données mais cela a vite été corrigé grâce aux outils *valgrind* et *gdb*. En effet, la libération de mémoire était plus importante que ce qui était alloué lors de la suppression de la liste et était donc la cause des plantages.

Nous avons également eu un problème avec les fonctions permettant de supprimer et d'afficher une liste d'enregistrements car elles étaient de type récursif et avec 449 fichiers de données, la taille de la liste était trop importante pour que tous les appels de fonctions soient placés sur le *stack*. Nous avons par conséquent pris la décision d'utiliser des boucles classiques pour ces fonctions.

Le fichier *.clang-format* était aussi la cause de soucis car certaines machines ne disposent que de la version 7 du programme et comme le fichier avait été initialement conçu pour la version 11, celui-ci n'était pas compatible.

6. Annexes

Les sources ci-dessous ont été utilisées dans la réalisation de ce projet :

- Instructions fournies en annexes du sujet.
- Conception du fichier .clang-format
<https://clang.llvm.org/docs/ClangFormatStyleOptions.html>
- Conception d'un Makefile
<https://www.youtube.com/watch?v=-riHEHGP2DU>
- Aide sur les bibliothèques C
<http://www.cplusplus.com/reference/>
- Fonctionnement de la fonction `strtok()`
https://www.tutorialspoint.com/c_standard_library/c_function_strtok.htm