Mini-projet C ANSI : `extraire` *ver. 1.0*

Les fichiers complémentaires, les mises à jour et une éventuelle foire aux questions seront disponibles sous *Celene*.

Le but de ce projet est de faire un premier pas vers la réalisation d'un SGBD en ligne de commande : réaliser un programme qui fasse une jointure élémentaire entre deux tables contenues dans des fichiers. La syntaxe est la suivante :

`extraire <champ>+ de <fichier1> <fichier2> avec <champ2>=<champ2>`

par exemple, la ligne de commande

`extraire a.1 b.1 a.2 b.3 de sport.table repas.table avec a.1=b.1`

avec les contenus des fichiers suivants :

sport.table

```
bob|natation
pierre | judo
paul | natation
jacques | judo
```

repas.table

```
alice | poisson | yourt
paul | viande | yaourt
paul | soupe | et au lit
paul | soupe | yaourt
jacques | poisson | fruit
pierre | viande | fruit
```

doit produire, sur la sortie standard :

```
pierre|pierre|judo|fruit|
paul|paul|natation|yaourt|
paul|paul|natation|et au lit|
paul|paul|natation|yaourt|
jacques|jacques|judo|fruit|
```

Les champs retenus sont, dans l'ordre, le premier champ du premier fichier (`a.1`) puis le premier champ du second fichier (`b.1`) puis le deuxième champ du premier fichier (`b.2`) puis... La jointure se fait sur les deux tables avec l'égalité entre les champs `a.1` et `b.1` .

L'ordre de sortie est celui du premier fichier puis du second.

Le projet est présenté en version basique puis des extensions sont proposées. Le travail demandé est indiqué ensuite. À la fin du document se trouvent quelques conseils et mises en garde.

1 Version basique

S'il n'est pas demandé d'implanter d'algorithme optimal, le programme doit néanmoins être relativement efficace.

1.1 Analyser la ligne de commande

Il faut analyser les arguments et ranger la requête dans une structure de donnée *ad-hoc*. Cette structure doit être présentée (dont graphiquement) dans le compte-rendu (CR). Expliquer (dans le CR) comment est traité le nombre non borné de champs (pour l'analyse comme pour la génération de la sortie).

Tout ce qui touche à l'analyse des arguments, au stockage de la requête et à sa manipulation doit être réuni dans un module `requete`.

1.2 Tables en mémoire

On suppose les deux fichiers suffisamment petits pour tenir complètement en mémoire. Pour simplifier, on suppose que chaque case a, au plus, une taille de 50 caractères. Le nombre de ligne n'est pas connu à priori et n'est pas borné. La structure permettant de contenir en mémoire une table doit être présentée dans le CR.

Tout ce qui touche à la lecture et à l'interrogation des tables doit être réuni dans un module `table`.

1.3 Jointure

Elle sera faite dans le fichier principal (elle peut même être dans le `main`) : `extraire.c`.

1.4 Structure de `file`

Vous *devez* implanter une structure *générique* de `file`, elle pourra vous être utile à plusieurs endroits. La structure de données utilisée ainsi que le fonctionnement d'une primitive (non triviale) doivent être présentées dans le CR.

L'entête du module (`file.h`) est imposé ainsi. Le module peut être testé par `make file_test` grâce au fichier (`file_test.c`).

2 Extensions

Dans chaque cas, il faut expliquer (dans le CR) l'algorithme et/ou la structure de données utilisé et dans quelle mesure cela demande ou non de compléter les autres modules.

2.1 Cases de taille quelconque

Faire en sorte qu'il n'y ait plus de limite sur la taille d'une case (on pourrait avoir des tailles de 1 Mo par exemple) tout en conservant l'hypothèse que les fichiers tiennent en mémoire.

2.2 Tri

Ajouter une option de tri (alphabétique, même lorsque ces données sont des chiffres) selon un champ (on se limite aux champs qui seront affichés). La nouvelle syntaxe est alors :
`extraire <champ>+ de <fichier1> <fichier2> avec <champ2>=<champ2> ordre <champ>`

2.3 Conditions d'autre nature

Permettre l'utilisation de conditions du type <champ2> < <champ2> (on utilisera < pour l'ordre alphabétique et <. pour l'ordre sur les nombres). Il n'y a toujours qu'une condition.

2.4 Conditions multiples

Permettre l'utilisation de plusieurs conditions simultanément. La nouvelle syntaxe est alors :
`extraire <champ>+ de <fichier1> <fichier2> avec <champ2>=<champ2>+`

2.5 Fichiers multiples

Permettre l'utilisation de plusieurs fichiers simultanément avec plusieurs conditions. La nouvelle syntaxe est alors :
`extraire <champ>+ de <fichier>+ avec < <champ2>=<champ2> >+`

2.6 Sans répétition

Ajouter une option permettant de ne plus répéter la moindre ligne (en sortie). Attention l'ordre de sortie doit être respecté, chaque ligne doit apparaître à sa première occurrence. Sur l'exemple introductif, la quatrième ligne n'apparaîtrait plus. La nouvelle syntaxe est alors :
`extraire <champ>+ de <fichier1> <fichier2> avec <champ2>=<champ2> [unique]`

2.7 Algorithmique

Comment rendre la jointure efficace ? Comment ôter toutes répétitions (comme demandé en 2.6) de manière efficace ? Cette extension ne demande pas d'implantation dans le code, elle est purement algorithmique et doit être développé dans le CR.

3 Travail demandé (par groupe de deux ou trois étudiants)

Vous devrez envoyer par courriel un fichier nommé `PASD_mini-projet.tzg` (archive `tar -czf` qui peut être engendré par `make archive`) contenant :

- tous les fichiers sources (`.h` et `.c`) ainsi que le `Makefile`, et
- un document `compte-rendu.pdf` d'au maximum 5 pages (+1 pour l'extension algorithmique).

Ce mail doit être *envoyé d'un compte étudiant* (`@etu.univ-orleans.fr`) au responsable du module (`jerome.durand-lose@lifo.univ-orleans.fr`). Aucune autre forme de remise ne sera acceptée.

Les projets seront notés en fonction du nombre de participants. Pour les groupes de deux étudiants, il n'est pas demandé d'implanter d'extension. Les groupes de trois étudiants doivent implanter au moins une extension.

Vous pouvez utiliser toutes les bibliothèques standard du C ANSI (celles qui n'ont pas besoin d'être indiquées à `gcc` par des `-l`). Aucune autre bibliothèque ne doit être utilisée.

Exceptionnellement, le projet pourrait être fait seul, mais cela doit être qu'après acceptation d'une *demande justifiée* auprès du responsable du module par courriel. Une extension est alors demandée.

3.1 Code

Le code contient au moins les fichiers suivants :

- `Makefile` à la base celui fourni (sous Célène et par mail). Vous pouvez y ajouter des dépendances et des modules [1] ;
- `file.c` pour le module implantant les `file` ;
- les `.h` et `.c` pour les modules `requete` et `table` ;
- `extraire.c`, le programme principal.

Il est possible d'ajouter d'autres fichiers en précisant lesquels et pourquoi (dans le `Makefile` et le CR).

Chaque fichier source doit contenir le nom des auteurs en copyright dans les commentaires. Il doit également contenir tous les commentaires nécessaires. Le code doit être lisible.

Le projet est en C ANSI pur, il doit compiler avec `gcc` et les arguments `-ansi -Wall -Wextra -pedantic` sans aucun *message d'alerte* (et encore moins d'erreur) comme prévu dans le `Makefile`.

Il ne doit y avoir *aucune constante magique* (*i.e.* seuls 0, 1, et -1 peuvent apparaître dans votre code en dehors des `#define`).

Il ne doit y avoir absolument aucune fuite mémoire (et encore moins de `segmentation fault`) et toutes les mémoires allouées doivent être rendues avant la fin de l'exécution du programme. `make memoire` permet de tester ceci.

3.2 Compte-rendu (CR)

Il doit comporter les noms des membres de l'équipe, répondre aux questions du sujet et préciser les éventuelles extensions considérées, le travail réalisé par chacun et ses limites. Le compte-rendu doit indiquer les difficultés rencontrées (et leurs solutions ou absence de). Il indique également ce qui a pu être appris / acquis / découvert durant le projet.

1. Il est bien entendu interdit de toucher aux options de compilation forçant le C ANSI.

3.3 Calendrier

Ce mini-projet est fait pour être fait rapidement. La *durée très limitée* fait partie de l'exercice.

Début du projet :	lundi 30 septembre
Remise blanche :	lundi 21 octobre (minuit)
Retour sur remise blanche :	jeudi 24 octobre
remise finale :	lundi 28 octobre (minuit)

Il est prévu une *remise blanche* à mi-parcours. Elle n'est pas obligatoire et n'entre pas dans l'évaluation du projet. Elle permet d'avoir un retour succinct sur ce qui aura été envoyé et donc de corriger pour la remise finale.

La notation pourrait être individuelle. Il n'est pas prévu pour l'instant de soutenance de projet. Mais cela pourrait être le cas, en particulier en cas de *zones d'ombre*.

4 Conseils

Cet énoncé est un cahier des charges *strict*, toute déviation sera pénalisée, d'autant plus qu'une partie de la correction sera automatisée.

Il faut commencer rapidement et ne pas perdre une semaine à former des groupes. La formation des groupes est de la responsabilité des étudiants.

La meilleure organisation est de considérer la remise blanche comme la remise finale. Si la date est tenue, cela permet d'avoir un retour et de perfectionner pour la remise finale. Si la date n'est pas tenue, la seule conséquence est que l'on se prive d'un retour, d'une chance d'identifier les problèmes et de les corriger (surtout si ce sont des problèmes comme un mauvais nom de fichier, une archive `.tgz` corrompue, un mauvais compte pour l'envoi de l'archive... ou autres qui sont des « non remises »).

Faire ses propres fichiers (C ou données) pour tester les différents modules et les utiliser souvent. Ils peuvent être joints au code, le CR précisant leur utilité et utilisation.

Pour faire de la généricité, commencer par une version non générique parfaite (et bien la sauvegarder) sur un type non trivial. Ensuite voir ce qui est particulier et le passer progressivement en argument / paramètre jusqu'à pouvoir enlever toute référence au type dont on s'abstrait. Chaque fois qu'une chose est abstraite, refaire tous les tests.

Faire très attention aux allocations et aux désallocations de mémoire.

Si vous n'arrivez pas à boucler le projet, rendez-le quand même en indiquant (dans le compte-rendu) ce qui a été fait et ce qui ne l'a pas été. De même *si tout ne marche pas parfaitement*, rendez le projet en indiquant ce qui coince dans le CR.

Il vaut mieux un projet *honnête* qu'un projet *dopé*. Le but est la maîtrise de la programmation en C, en particulier des pointeurs, pas de faire une nouveau moteur de SGBD.

4.1 Pour ceux qui auraient des doutes ou des « tentations »

Dans le cas de l'utilisation d'un autre langage que le C ANSI, ou du non respect du cahier des charges, le mini-projet sera « non remis », *i.e.* noté zéro.

Les « partages » de code entre groupe et les « emprunts » de code sur internet seront à expliquer et justifier dans le compte-rendu (ou à défaut en commission de discipline).