Mini-projet C ANSI : manipulation de XML allégé *ver. 1*

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.

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

Le but de ce projet est de réaliser un premier outil capable de vérifier et de manipuler un document XMLight. Sa syntaxe générale est :

`ixemelle <ordre>* <fichier>.xmlight`

Par exemple,

`ixemelle data.xmlight`

vérifie que le fichier `data.xmlight` est du xmlight valide et affiche ensuite le contenu lu reformaté.

`ixemelle cut panier data.xmlight`

enlève des données lues dans le fichier `data.xmlight` tous les sous-arbres correspondants à la balise `panier`.

`ixemelle cut 'panier -> pommes <-- [commestible="vrai"]' data.xmlight`

enlève tous les sous-arbres correspondant à la balise `panier` ayant un fils `pomme` et un ancêtre ayant un attribut `commestible` égal à `vrai`.

À chaque fois, le résultat est envoyé sur la sortie standard en xmlight.

Voici une entrée `xmlight` (bien formatée), à gauche. À droite se trouve le résultat affiché par `./ixemelle cut 'coissant' cut '[nombre="2"]' cut '-> brocoli' acheter.xmlight`

```
<boulangerie ferme="lundi">
 <croissant nbr="4"/>
 <baguette nbr="2"/>
</boulangerie>
<marche>
 <panier numero="1">
  <pomme kg="3"/>
  <poires kg="5"/>
 </panier>
 <panier numero="2">
  <poireaux bottes="5"/>
  <choux type="blanc" nombre="2"/>
  <choux type="vert" nombre="1"/>
 </panier>
 <panier numero="3">
  <brocoli/>
 </panier>
</marche>
```

```
<boulangerie ferme="lundi">
 <croissant nbr="4"/>
 <baguette nbr="2"/>
</boulangerie>
<marche>
 <panier numero="1">
  <pomme kg="3"/>
  <poires kg="5"/>
 </panier>
 <panier numero="2">
  <poireaux bottes="5"/>
  <choux type="vert" nombre="1"/>
 </panier>
</marche>
```

1 Format XML allégé

Il s'agit de XML, mais avec les restrictions suivantes :

— il n'y a pas de texte entre les balises (sinon des caractères espace, tabulation, retour à la ligne...) ;

— une balise ouvrante est de la forme <<nom> <attribut>*>.
Le nom est une chaîne de caractères (d'au moins 1 caractère et) d'au plus 15 caractères ; elle est composée uniquement de lettres. Les noms sont insensibles à la casse (c.-à-d. `Ac` et `aC` sont la même clé). Le nombre d'attributs n'est pas limité ;

— ces attributs sont sous la forme <clé>="<valeur>".
<clé> est une chaîne de caractères (d'au moins 1 caractère et) d'au plus 10 caractères ; elle est composée uniquement de lettres. Les clés sont insensibles à la casse.
<valeur> est une chaîne de caractères d'au plus 50 caractères où le caractère '"' est protégé par une barre oblique inverse (\") [1],
— une balise fermante est de la forme </<nom>> ;
— les balises ouvrantes et fermantes doivent bien entendu se correspondre comme des parenthèses ;
— il peut ne pas y avoir de balise fermante, mais la balise doit alors se terminer par />.

2 Version basique

Quelque soit le traitement demandé, le programme devra toujours commencer par vérifier et charger en mémoire toute l'information du fichier xmlight. Une fois les données chargées, les ordres sont exécutés. Le résultat est ensuite affiché.

2.1 Structure de forêt de listes

Vous devez commencer par implanter la structure *générique* `list_forest`. Par la suite, elle sera particularisée pour stocker du xmlight.

Il s'agit d'une structure d'arbre d'arité non bornée (le nombre de fils n'est pas limité). Chaque nœud permet accéder au *père*, au *frère suivant* (à droite) et à son *premier fils* (à gauche). Cette structure de données est connue comme, p.e., Left-child right-sibling binary tree et Rooted trees with unbounded branching.

Le nœud racine est la racine du premier arbre, ses frères sont les racines des autres arbres. C'est pour cela que l'on parle de forêt.

Chaque nœud contient les données suivantes :
— une valeur (générique qui servira à stocker le nom), et
— une liste d'éléments (génériques qui serviront à stocker les attributs des balises).

La gestion de la liste est totalement intégrée à la structure (ce n'est pas une liste à part). On se balade dans la structure (forêt et listes) au travers d'une structure `list_forest_position`.

L'en-tête du module (`list_forest.h`) est imposé. Le module peut être testé grâce au fichier (`file_list_forest.c`) avec `make t_list_forest`. Bien les regarder pour comprendre le fonctionnement et donc ce qu'il faut implanter.

La structure de données utilisée ainsi que le fonctionnement d'une primitive (non triviale) doivent être présentés dans le compte-rendu (CR).

2.2 Initialisation de la forêt pour xmlight

Il s'agit d'écrire le module `ixmlight`.

2.3 Lecture du xmlight d'un fichier

Pour écrire le module `ixmlight_io`, il faut prévoir des fonctions pour la lecture d'un nom, d'une clé, de la valeur associée, d'un attribut... En cas d'erreur de syntaxe, un code d'erreur doit être retourné. La structure est alors laissée partiellement remplie. Pour tester, commencer à écrire `ixmlight_main.c` : on se contente de lire puis d'afficher la structure.

1. Attention elle peut contenir tout type de caractères, y compris des retours à la ligne, des tabulations...

2.4 Critère

Écrire le module `ixmlight_criterion`. La lecture à partir d'une chaîne de caractères peut être testée par `make t_criterion`.

La structure de données utilisée ainsi que le fonctionnement d'une primitive (non triviale) doivent être présentés dans le CR.

2.5 Suppression de nœuds

Écrire le module `ixmlight_cut`. Pour le tester, compléter l'écriture de `ixmlight_main.c` afin qu'il détecte et lance les `cut`.

Le fonctionnement doit être présentées dans le CR.

3 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.

Il faut également *fournir des fichiers de tests* (ordre, donnée, résultat attendu).

3.1 Ordre d'extraction

Il s'agit de faire une commande qui regroupe tous les nœuds (avec leurs sous-nœuds) vérifiant un critère. La syntaxe est de la forme :

```
./ixemelle get '[nombre="2"]' acheter.xmlight
```

3.2 Gérer le texte entre les balises

On suppose qu'il y a au plus 900 caractères entre deux balises. Les espaces en début et fin de texte sont supprimés. Il est affiché entre les balises en sortie.

3.3 Critères sur attributs

Il s'agit d'une part d'ajouter des tests du type : `[nombre != "2"]` et `[nombre <= "2"]` (avec vérification qu'il s'agit bien de nombres pour les comparaisons) et d'autre part de permettre d'en avoir plusieurs en même temps séparés par des points-virgules : `[nombre != "2" ; nombre < "9"]`. Ils doivent alors tous être vérifiés (ici `nombre` n'est pas `"2"` et est inférieur à `"9"`).

3.4 Conditions de parenté multiples et relatives

Il s'agit de permettre d'une part des critères du type : `-> pain -> fromage` (avoir un fils `pain` et un fils `fromage`) mais aussi du type `-> ( pain --> ( chevre <-- fromage ) )` : avoir un fils `pain` ayant un descendant `chevre`, celui-ci ayant un ascendant `fromage`.

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

Les fichiers en-tête fournis (`.h`) sont à respecter *scrupuleusement*. S'il n'est pas demandé d'implanter d'algorithme optimal, le programme doit néanmoins être relativement efficace.

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`,

— un document `compte-rendu.pdf` d'au maximum 5 pages, et
— tous les fichiers annexes pour tester d'éventuelles extensions.

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

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 standards du C ANSI (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 *uniquement après* acceptation d'une *demande justifiée* auprès du responsable du module par courriel. Une extension doit alors être réalisée.

4.1 Code

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. Ceci peut être testé avec `make memoire`.

4.2 Compte-rendu (CR)

Il doit comporter les noms des membres de l'équipe, répondre aux questions du sujet, préciser les éventuelles extensions considérées, les limites de ce qui a été réalisé et le *travail réalisé par chacun.*

Le compte-rendu doit indiquer les difficultés rencontrées (et leur solution ou absence de). Il indique également ce qui a pu être appris / acquis / découvert durant le projet.

4.3 Calendrier

Ce mini-projet est fait pour être fait rapidement. La restriction du temps fait partie de l'exercice.

Début du projet	: lundi 29 septembre
Remise blanche	: lundi 20 octobre (minuit)
Retour sur remise blanche	: jeudi 23 octobre
Remise finale	: lundi 27 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 pourra être individuelle. Il n'est pas prévu pour l'instant de soutenance de projet. Mais cela pourra être le cas, en particulier en cas de *zones d'ombre.*

5 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 pour manipuler du XML. La maîtrise de tous les détails pour la bonne réalisation de la structure `formest_list` est déjà un jalon important.

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