Travail pratique #3
INF3135

Automne 2007

- Peut être fait seul(e) ou en équipe de deux (auquel cas les deux étudiants doivent alors provenir *du même groupe-cours*).
- Date de remise (électronique) :

Groupe 20	11 décembre 2007	18h00
Groupe 30	12 décembre 2007	8h30

Aucun travail ne sera accepté après la date suivante :

Groupe 20	18 décembre 2007	18h00
Groupe 30	19 décembre 2007	8h30

1 Introduction

Pour ce travail pratique, vous allez devoir modifier du code existant, donc faire de la maintenance.

Un logiciel de gestion de prêts de livres pour une bibliothèque personnelle a été développé. Bien que fonctionnel, ce logiciel requiert que des améliorations soient apportées. Il s'agit donc de faire de la *maintenance préventive et perfective* et vous allez participer à ce travail de maintenance.

Dans un premier temps, vous devrez obtenir la version courante du code source du logiciel ainsi que la documentation qui décrit son fonctionnement. Ensuite, vous devrez comprendre le fonctionnement du logiciel, tant du programme lui-même que des tests. Vous serez alors prêt à modifier le code et les tests, puis à produire une nouvelle version du logiciel. Finalement, une fois vos modifications complétées, vous devrez remettre l'ensemble du code modifié à votre enseignant.

2 Installation

La première étape pour la réalisation de ce travail est d'installer une copie d'un arbre CVS dans votre compte. Pour ce faire, il vous suffit d'exécuter, à partir de la machine `arabica`, la commande suivante (sans argument) dans le répertoire où vous voulez que le nouveau sous-répertoire soit créé :

```
~tremblay/public_html/INF3135/TP3/obtenir-code-source
```

Pour comprendre le rôle et la structure du logiciel de gestion de prêts de livres, vous devez consulter deux «documents» :

1. Un premier document qui décrit d'où origine ce logiciel, de quelle façon il a évolué, quelle est sa structure actuelle, comment il est documenté, comment il est testé, etc.

   ```
   http://www.labunix.uqam.ca/~tremblay/INF3135/Biblio/systeme-gestion-livres.pdf
   ```

   Les principaux chapitres de ce document traitent des points suivants :

   1. Introduction : les origines du système de gestion de prêts de livres
   2. État de la première version du logiciel
   3. État de la version actuelle

4. Travaux futurs

A. Les spécifications d'origine

B. Stratégie de tests pour la vérification du bon fonctionnement et de la non régression du logiciel

C. Exemple de résultat produit avec `DOC++` pour les opérations du module `gererArguments`

2. La documentation en ligne, en format `DOC++`, qui décrit les principaux modules du logiciel :

   `http://www.labunix.uqam.ca/~tremblay/INF3135/Biblio/systemeGestionLivres`

À l'aide de ces documents, il vous faudra comprendre le fonctionnement du logiciel, la structure du code contenu dans l'arbre CVS, la façon dont les tests sont réalisés, comment les modules et fonctions doivent être documentés, etc.

Suggestion : *Assurez-vous dès le départ du bon fonctionnement de votre arbre CVS en utilisant quelques commandes CVS. Vérifiez aussi que le* `Makefile` *est bien configuré en exécutant les commandes «*`make`*», «*`make tests`*» et «*`make testsu`*».*

3 Modifications à effectuer

Le logiciel permet la gestion d'une base de données de livres prêtés. Plus précisément, les données de cette BD représentent uniquement les livres ayant été prêtés, et non l'ensemble des livres d'une bibliothèque. Un certain nombre de commandes différentes sont disponibles et sont décrites dans le document mentionné plus haut. Votre travail va consister à améliorer certaines parties du code, tout en vous assurant de ne pas modifier son comportement — c'est-à-dire qu'il doit continuer à satisfaire l'ensemble des tests, y compris certains tests additionels que vous ajouterez.

Plus précisément, voici la description des modifications et améliorations que vous devez apporter au logiciel :

1. Le compilateur `gcc` possède une option `-O` qui permet d'effectuer une *analyse de flux de données*. Entre autres, une telle analyse permet de détecter les cas où une variable *pourrait être utilisée sans avoir été initialisée*.

   Modifiez le fichier `Makefile` pour ajouter cette option de compilation et faites ensuite les modifications requises pour corriger les cas signalés par le compilateur — c'est-à-dire les modifications requises pour supprimer les avertissements générés par le compilateur.

2. Toutes les commandes, sauf la commande `lister`, sont mises en oeuvre directement par le logiciel `biblio`. Par contre, la commande `lister` est plutôt réalisée à l'aide d'un *script* (*shell script*). Vous allez devoir mettre en oeuvre, en code 'C' intégré à l'ensemble du logiciel, cette commande.

   Voici un bref résumé des tâches à accomplir :

   - Modifiez les fichiers de tests de niveau système (du répertoire `Tests`, exécutés avec la commande `make tests`). Ces tests sont actuellement définis pour utiliser la commande «`./lister`» définie via le `shell script`. Il vous faut modifier ces fichiers de tests de façon à ce qu'ils utilisent plutôt la commande «`biblio lister`».

     Dans l'état actuel du logiciel qui vous est fourni, la commande «`biblio lister`» n'est pas reconnue par le programme principal — donc signale une erreur. Après la modification des jeux d'essai, il y aura alors un certain nombre de tests qui ne fonctionneront pas. Votre travail consiste donc ensuite à modifier le code pour assurer le bon fonctionnement de ces tests.

   - Écrire le code pour la commande «`biblio lister`».

     Plus spécifiquement, dans sa forme de base, la commande `biblio lister` ne reçoit aucun argument. Elle parcourt la base de données et produit, sur la sortie standard, la liste de tous les livres prêtés — rien n'est affiché si la base de données est vide, c'est-à-dire si aucun livre n'est prêté.

- Définir des *tests unitaires* pour vérifier, de façon plus détaillée, que la (les) routine(s) introduite(s) pour réaliser la nouvelle commande fonctionne(nt) correctement — les tests de niveau système sont minimaux et ne couvrent pas nécessairement toutes les possibilités. Les tests unitaires que vous devez développer doivent être dans le style des tests de ceux déjà présents pour certains des autres modules ou routines — voir la commande «`make utests`» et le module `MiniCUnit`.
- Documenter, dans le style `DOC++`, les modifications que vous aurez effectuées aux fichiers d'interface (fichiers `.h`).

Quelques informations et indices :

- Le résultat produit par `biblio lister` doit être trié, tout d'abord selon le premier champ (nom de l'emprunteur), puis le champ suivant (auteur(s)), et finalement sur le dernier champ (titre du document). Par contre, ce tri peut ne s'effectuer *que sur la partie effectivement affichée.*
- Pour le tri, vous n'avez pas à définir votre propre procédure de tri. Vous devez plutôt utiliser la procédure `qsort` définie dans la bibliothèque standard (`<stdlib.h>`) — pour plus d'informations, voir «`man qsort`».

Vos modifications doivent utiliser à bon escient les modules déjà existants. Vous devrez aussi vous adapter au style de programmation utilisé dans l'ensemble du code qui vous est fourni.

4 Remise

Pour la remise de votre solution, il vous suffira d'utiliser la commande suivante, que vous devrez exécuter *à partir de la machine* `arabica` *et à partir de votre répertoire* `TP3-INF3135`, commmande qui aura pour effet de créer puis rendre un fichier d'archive :

```
~tremblay/public_html/INF3135/TP3/remettre ekharraz nomEquipe
OU
~tremblay/public_html/INF3135/TP3/remettre chieze nomEquipe
```

Où :

- *`nomEquipe`* est l'un ou l'autre des identifiants suivants, selon que vous travaillez seul ou en équipe de deux :
  - `CodePermanent`
  - `CodePermanent1,CodePermanent2`

Le travail doit être remis au plus tard à la date indiquée au début du document. Tout travail remis après cette date sera considéré en retard (pénalité de 10 % par jour). Aucun travail ne sera accepté après la date indiquée au début du document.

Liste de vérifications

- Vous avez écrit vos noms et vos codes permanents au début des divers fichiers sources que vous avez modifiés ou ajoutés.
- Votre programme compile correctement lorsque vous lancez la commande `make`.

5 Évaluation

Votre solution à ce travail sera évaluée selon les critères suivants :

- Fonctionnalité (8 pts) : Compilation correcte et bon fonctionnement du code (tels que jugés selon les tests systèmes déjà fournis...ou les autres tests qui pourraient être définis par l'enseignant pour évaluer le bon fonctionnement).
- Bonne programmation (2 pts) : Disposition du code, choix d'identificateurs significatifs. Gestion des cas d'erreur, utilisation judicieuse d'assertions et de programmation défensive.
- Tests unitaires (3 pts) : Qualité (par exemple, couverture) des tests unitaires définis pour tester vos nouvelles routines.
- Intégration avec le code existant (2 pts) : Bonne utilisation des routines existantes. Style de programmation qui respecte le code déjà présent. Mise à jour correcte des fichiers de tests de niveau système.