Université Bordeaux 1, Sciences et technologies
UFR Informatique et Mathématiques

Licence Sciences et Technologies, mention Informatique
Session de janvier 2004

Programmation Fonctionnelle est Symbolique
Devoir maison dû le 9 janvier à 12h00
Version préliminaire

Ce devoir est à faire en binômes, ou exceptionnellement seul. Chaque étudiant peut participer à au plus un binôme et peut contribuer au code d'au plus une copie rendue.

Les deux membres d'un binôme collaborent sur le devoir, mais ne doivent pas discuter des questions relatives au devoir avec un membre d'un autre binôme. Les questions de nature générale doivent être posées par courrier électronique à strandh@labri.fr. La réponse sera envoyée (également par courrier électronique) à l'expéditeur normalement au plus 24h après réception de la question. Si la question est jugée d'intérêt général, la question et la réponse seront postées dans lstinfo. Veuillez consulter lstinfo avant d'envoyer une question.

Vous êtes encouragés à consulter tout document (livre, article, site web, etc) afin de vous aider à produire votre résultat. Les règles habituelles s'appliquent, à savoir que toute inclusion de code trouvé ailleurs doit être déclarée en tant que telle. L'acte de faire passer du code écrit par une autre personne comme étant le vôtre s'appelle "plagiat" et est formellement interdit.

Toute collaboration non permise, plagiat détecté ou soupçonné sera rapportée au président de l'université. La pénalité possible inclut l'exclusion provisoire ou définitive de l'université.

La copie rendue doit être étiquetée par *deux numéros d'anonymat*, ou (exceptionnellement) un seul. Toute copie rendue portant plus de deux numéros d'anonymat sera considérée comme non valide. Le même barème sera appliqué à une copie portant deux ou un seul numéro. Vous avez donc intérêt à travailler en binômes.

Le travail rendu sera jugé selon les critères suivants :

- Clarté et lisibilité du code. Le code doit être le plus clair et le plus lisible possible. Utiliser la construction la plus spécifique applicable dans chaque cas.
- Respect de la culture partagée. Le code doit respecter les traditions de Lisp, y compris indentation, espacement, noms de variables, formats de commentaire et utilisation d'idiomes. Attention à ne pas utiliser l'indentation pour Emacs Lisp à la place de l'indentation pour Common Lisp. Il sera parfois nécessaire de rajouter des règles d'indentation dans Emacs, en particulier concernant des macros éventuelles réalisées.
- La taille du code. Le code doit être le plus compact possible (en nombre d'expressions) toute en respectant la culture partagée. La duplication de code doit être évitée autant que possible grâce à l'utilisation d'abstractions sous la forme d'abstractions de données (classes, structures, etc), d'abstractions de contrôle (fonctions, méthodes, etc) et d'abstractions de syntaxe (macros).
- Performance. Le code doit utiliser des algorithmes et des structures de données performants, sauf si la taille du problème peut être considérée comme faible et que l'algorithme ou la structure de donnée moins efficace est considérablement plus lisible.
- Modularité. Le code doit, autant que possible, être séparé en modules indépendants (dans des fichiers différents ; il n'est pas nécessaire d'utiliser les paquetages).
- Extensibilité. Le code doit permettre des extensions futures, de préférence sans modification du code fourni, et surtout sans qu'une application existante utilisant le code ne doive être modifiée.

Bon courage !

On souhaite écrire une partie d'un programme destiné à la gestion de scolarité d'une université. La structure du programme doit refléter plusieurs phases d'améliorations.

1 Fonctionnalités de base

Le programme doit permettre l'*inscription* d'un ensemble d'*étudiants* dans un ensemble d'*unités d'enseignement*.

La représentation d'un étudiant doit contenir les informations habituelles comme le nom, l'adresse, le numéro d'étudiant, la mention (par exemple "informatique") à laquelle il est inscrit, etc.

La représentation d'une unité d'enseignement doit contenir des informations comme le département qui l'organise, le nombre de crédits ECTS, etc.

À partir d'une *mention* il doit être possible de trouver le directeur, l'ensemble des étudiants inscrits à cette mention, ainsi qu'une liste exhaustive des unités d'enseignement suivies par les étudiants de la mention.

Le logiciel doit permettre l'inscription d'un étudiant à une unité d'enseignement. Le logiciel doit permettre une telle inscription seulement si l'UE fait partie de la liste des UE déjà suivies par les étudiants de la mention. Pour pouvoir effectuer l'inscription, l'utilisateur doit préalablement rajouter l'UE à la liste.

Le logiciel doit contenir les fonctionnalités suivantes :

– il doit permettre l'inscription d'un étudiant dans une mention ;
– il doit permettre l'inscription d'un étudiants déjà inscrit dans une mention dans une unité d'enseignement ;
– étant donné le nom d'une unité d'enseignement, il doit pouvoir récupérer la liste des étudiants qui suivent l'unité ;
– étant donné le nom d'une unité d'enseignement et le nom d'une mention, il doit pouvoir récupérer la liste des étudiants de la men-

tion qui suivent l'unité ;
- étant donné le nom d'un étudiant, il doit pouvoir donner la liste des unités auxquelles est inscrit l'étudiant ;
- il doit pouvoir donner la liste de toutes les unités d'enseignement ayant un nombre d'étudiants inscrits inférieur à un nombre donné.

2 Évolution du logiciel

On imagine une évolution des fonctionnalités du logiciel de la section précédente. Pour des raisons de maintenance, on ne souhaite pas toucher aux fonctionnalités de base. On souhaite même que les fichiers contenant les fonctionnalités de base ne changent pas. Toutes les fonctionnalités de cette section doivent donc être situées physiquement dans des fichiers à part.

Un étudiant n'ayant pas le droit d'être inscrit à plus de 30 crédits, on souhaite empêcher toute inscription qui entrainerait la violation de cette règle. Toute tentative d'une telle inscription doit signaler une condition (lire la HyperSpec pour plus de détails sur les conditions).

On souhaite également introduire un nouveau type d'étudiant appelé *salarié*. L'inscription d'un salarié doit également vérifier le nombre de crédits maximum, sauf que pour un salarié, le seuil est de 18 crédits au lieu de 30.

Le logiciel doit également vérifier avant une inscription dans une mention que l'étudiant n'est pas déjà inscrit dans une autre mention.

3 Encore une évolution

Une deuxième évolution du logiciel doit permettre à chaque unité d'enseignement d'avoir un emplois du temps. Pour simplifier, on imagine que chaque unité a un seul groupe d'étudiants et donc un ensemble unique de créneaux de cours, TD et TP.

L'inscription d'un étudiant dans une unité d'enseignement doit vérifier que l'emploi du temps de l'unité est compatible avec celui des autres unités d'enseignement auxquelles est inscrit l'étudiant.

De plus, le logiciel doit pouvoir fournir l'emploi du temps d'un étudiant quelconque.

4 À rendre

Vous devez :

– tester votre programme (et rendre le résultat des tests avec le programme) sur au moins 20 étudiants,
– choisir des cas qui permettent de tester la totalité du code,
– fournir le résultat des tests.

5 Documentation

Fournir *au plus une page* de texte expliquant ce que vous avez fait. Ne pas expliquer le code, mais les choix considérés mais non retenus (et pourquoi) concernant les représentations des données, les algorithmes choisis et l'organisation du programme. Expliquer également comment étendre votre programme selon les exigences indiquées ci-dessus.