Examen de programmation avancée

ENSIIE, semestre 2

mercredi 26 mars 2014

Durée : 1h45.
Tout document personnel autorisé (pas de prêt entre voisins). L'usage de la calculatrice ou de tout autre appareil électronique n'est pas autorisé.
Ce sujet comporte 5 exercices indépendants, qui peuvent être traités dans l'ordre voulu. Il contient 4 pages.
Le barème est donné à titre indicatif, il est susceptible d'être modifié. Le total est sur 20 points.

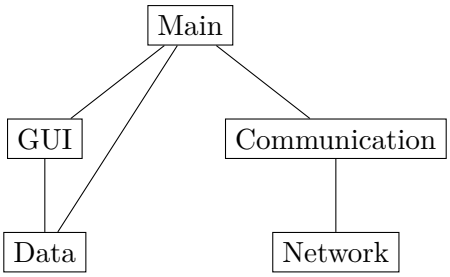
Exercice 1 : Modularité (2 points)

Quelles propriétés conséquences de la modularité (parmi séparation des tâches, création d'un espace de nom, réutilisabilité, encapsulation, abstraction et maintenabilité) permettent d'éviter les écueils suivants ? On indiquera le nom de la propriété et on justifiera comment elle résout le problème.

1. « Les autres programmeurs n'arrêtent pas d'utiliser les fonctions auxiliaires que j'ai écrites seulement pour moi, et ça casse mes beaux invariants. »
2. « Je suis à court d'imagination pour trouver des noms à mes fonctions qui soient différents des noms déjà existants pour d'autres structures de données. »
3. « À chaque fois que je change un petit bout de mon code, je suis obligé de modifier des tas d'autres trucs. »
4. « C'est la quarante-deuxième fois que je réécris mon implémentation des listes chaînées. »
5. « On programme tous dans le même fichier, et on échange par email pour savoir qui peut toucher à quoi. »
6. « Je ne comprends rien à l'implémentation de cette structure et du coup j'ai du mal à savoir comment l'utiliser. »

Exercice 2 : Compilation séparée (3 points)

On a choisi de découper un projet selon les modules suivants :

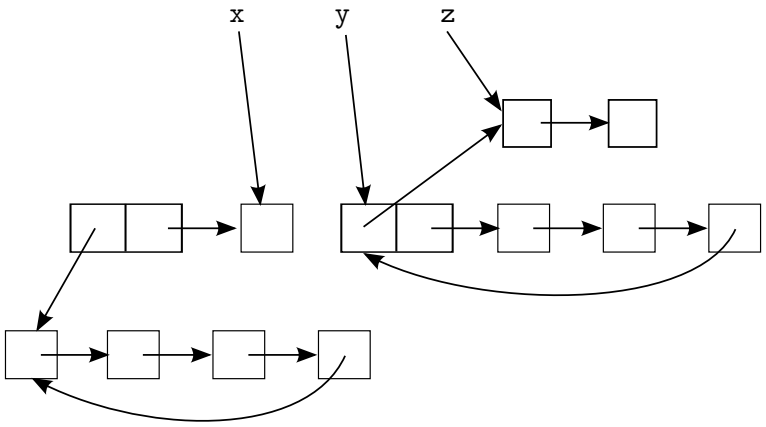


Un module m1 est relié par une arête vers le bas à un autre module m2 si m1 dépend de (l'interface de) m2. On suppose que chaque module sauf Main a une interface explicite, et que si elle existe, un module dépend de son interface. On pourra considérer au choix que les modules sont des fichiers C ou OCaml.

1. Quelle commande faut-il écrire manuellement pour compiler le module Main ? En OCaml, on supposera que les interfaces des modules sont déjà compilées.
2. En supposant que tous les modules sont compilés, quelle commande faut-il taper manuellement pour faire l'édition de liens ? (Attention à l'ordre des modules en OCaml !)
3. On modifie l'interface de Network. Quel(s) module(s) faut-il recompiler ?
4. Écrire le `Makefile` correspondant au projet. On n'oubliera pas d'écrire une cible pour produire l'exécutable final qu'on appellera `prog`.

Exercice 3 : Garbage collection (4 points)

On suppose qu'à un moment dans un programme on arrive à un état de la mémoire suivant :



Les variables du programme sont `x`, `y` et `z`.

1. Quels sont les parties accessibles depuis le programme ?
2. Appliquer un algorithme de GC de type comptage de références dessus. Que peut-on remarquer ?
3. Appliquer un algorithme de type « marquer et nettoyer ».
4. Appliquer un algorithme de GC copiant.

On suppose maintenant qu'après application du GC copiant, `y` est modifié et pointe maintenant vers une zone mémoire de taille 1 nouvellement allouée dont le contenu pointe vers la même zone que `z`.

5. Représenter le changement dans la mémoire, et appliquer à nouveau un algorithme de GC copiant.

Exercice 4 : Annuaire inversé (4 points)

On souhaite utiliser une structure de dictionnaire pour coder un annuaire inversé. Les clefs seront donc des numéros de téléphone, et les valeurs le nom du propriétaire du numéro.

1. On veut que les opérations de recherche en moyenne soient très rapides. Quelle structure de données parmi celles vues en cours vaut-il mieux utiliser ? Justifier.

Sans préjuger de la réponse à la question précédente, on décide d'utiliser des tables de hachage pour implémenter le dictionnaire.

2. On suppose qu'on a 100 entrées à stocker dans notre annuaire. Expliquer pourquoi il serait mauvais de choisir comme fonction de hachage la fonction qui retourne les deux premiers chiffres du numéro vus comme un entier entre 0 et 99.

   Rappel : en France, les numéros de téléphone sont composés de 10 chiffres. Le premier est 0, le second chiffre vaut 1, 2, 3, 4 ou 5 pour les fixes (selon la région), 6 et 7 pour les mobiles, et 8 pour les numéros spéciaux.

3. On utilise la fonction de hachage suivante : on fait la somme des chiffres du numéro, et on prend le reste de la division euclidienne de ce nombre par la taille de la table.

   On suppose maintenant qu'on a une table de hachage ouverte de taille 5. Représenter l'état de la table après l'insertion des associations suivantes :

Numéro	Nom
0123456789	Jean Dupont
0612152601	Jean Dupont
0276345291	Pierre Martin
0723762910	Jacques Dupond
0422901764	Martin Pierre

4. On utilise une table de hachage avec redimensionnement dynamique (en doublant la taille en cas de redimensionnement). Représenter l'état de la table précédente après une nouvelle insertion, celle de l'association de 0899876549 avec Arnaque Télécom.

Exercice 5 : Piles (7 points)

Vous pouvez faire cet exercice au choix en C ou en OCaml.
On considère une structure de pile avec les fonctions suivantes :

- `faire_vide` sans arguments qui renvoie une pile vide ;
- `est_vide` qui prend une pile et renvoie un booléen qui vaut vrai si la pile est vide ;
- `empiler` qui prend un entier, une pile, qui insère l'entier dans la pile et renvoie la nouvelle pile ;
- `dessus` qui prend une pile et qui renvoie l'élément en tête de la pile ;
- `depiler` qui prend une pile et qui renvoie la pile privée de sa tête.

1. Écrire l'interface du module pour les piles.

On implémente les piles à l'aide d'un tableau, d'un entier `tete` indiquant la position de l'élément en tête de la pile, et d'un entier `taille` indiquant la taille du tableau.

- la pile vide est un tableau de taille 1 avec la position `tete = -1` ;
- une pile est vide si `tete` vaut `-1` ;
- `dessus` retourne la case du tableau à la position `tete` ;
- `depiler` décrémente `tete` de `1` ;
- pour `empiler`, on incrémente `tete` ; si la nouvelle valeur est strictement inférieure à `taille`, on place l'élément à cette case, et on renvoie l'ancienne pile ; sinon, on crée un nouveau tableau de taille double, on recopie tous les éléments de l'ancien tableau dans ce tableau, et ajoute le nouvel élément dans la case du nouvel indice `tete` ; on renvoie une pile constituée du nouveau tableau, du nouvel indice `tete` et de la taille doublée.

2. Donner la définition concrète du type pour les piles.
3. Implémenter les fonctions de la façon décrite ci-dessus.
4. Donner la complexité des fonctions `dessus` et `depiler` en fonction de n le nombre d'élément dans la pile, en moyenne et dans le pire des cas.
5. Donner la complexité de la fonction `empiler` en fonction de n le nombre d'élément dans la pile dans le pire des cas.
6. On considère l'insertion successive à l'aide d'`empiler` de k éléments à partir d'une pile vide.
   a) Dessiner l'état de la pile au fur et à mesure des insertions pour $k = 5$.
   b) Quelle est la complexité totale de ces k insertions (pour k quelconque) ?
   c) En déduire la complexité en moyenne de `empiler`.
7. Quels sont les avantages et les inconvénients de cette implémentation par rapport à une implémentation à l'aide de liste chaînée ?