Université Bordeaux 1, Sciences et Technologies
UFR Mathématiques et Informatique

Maîtrise d'Informatique
Session de janvier 2000

**Techniques et Fondements de la Programmation (TFP)**
**partie "techniques"**
**le 28 janvier 2000**

Durée : trois heures (les deux parties)

Tous documents autorisés sauf photocopies de livres. Votre voisin n'est pas un document.

Rappel : Soyez bref et clair ! Personne n'est impressionné par la quantité !

Bon courage !

**Exercice 1. (7p)**

Une file de priorités est un conteneur dans lequel chaque élément est associé à une *priorité* (un entier positif). À l'insertion (opération `insert`) d'un élément dans la file, cette priorité est donnée explicitement. L'opération `dequeue_max` de suppression d'un élément renvoie et supprime l'un des éléments de la file ayant la plus grande priorité. Finalement l'opération `empty` renvoie une valeur indiquant si la file est vide.

Souvent, on peut supposer que la priorité maximale est relativement petite. Avec cette supposition, on peut représenter la file sous la forme d'un tableau dont la taille est égale à la priorité maximale. Si de plus, on se permet de donner la priorité maximale à la création de la file, ce tableau peut être de taille fixe pendant toute la durée de vie de la file.

Avec ces suppositions, écrire (en langage C) l'interface et l'implémentation d'une file de priorités. Vous pouvez supposer l'existence d'un type *liste* dont les opérations sont `cons(head, tail)`, `car(list)`, et `cdr(list)`.

**Exercice 2. (7p)**

Écrire une macro Common Lisp permettant la simulation d'une boucle `for` du langage C, y compris la possibilité d'interrompre la boucle avec la forme `(break)` et de passer immédiatement à l'itération suivante avec la forme `(continue)`.

Indication: créer un `block` à l'extérieur de la boucle à utiliser avec `(break)`, et un `block` à l'intérieur à utiliser avec `(continue)`, puis définir (avec `flet`) deux fonctions `break` et `continue` dont la fonction est d'effectuer un `goto` non local au `block` correspondant.

**Exercice 3. (6p)**

Dans un jeu de Pacman, un labyrinthe contient un ensemble de cellules. Une cellule peut être vide ou occupée par un ensemble d'objets de types différents. Les types d'objets sont :

`pastille` : les objets de ce type sont immobiles, et certaines sont des super pastilles.
`fantôme` : les objets de ce type sont mobiles.
`pacman` : les objets de ce type sont aussi mobiles.

Quand un objet $A$ ce déplace d'une cellule à une autre cellule déjà occupée par un ou plusieurs autres objets, il y a une `collision` entre l'objet $A$ et chaque autre objet de la cellule. Il n'y a jamais collision entre deux pastilles. La collision entre un fantôme et une pastille ou entre un fantôme et un autre fantôme ne fait rien. Une collision entre un pacman et une pastille a pour conséquence de supprimer la pastille du labyrinthe et d'augmenter la force du pacman. Si de plus, la pastille est une super pastille, alors, le pacman devient invincible. Une collision entre un pacman invincible et un fantôme a pour conséquence la suppression du fantôme. Si le pacman n'est pas invincible, alors c'est le pacman qui est supprimé.

Modéliser les types d'objets et les collision avec des classes, des fonctions génériques et des méthodes CLOS. Vous avez à votre disposition une fonction `delete-object` permettant de supprimer n'importe quel objet du labyrinthe. On suppose que d'autres fonctions du programme s'occupent de déplacer les objets, de détecter les collision, et d'appeler la fonction générique `collision` pour chaque collision détectée.