Université Bordeaux 1, Sciences et Technologies
UFR Informatique et Mathématiques

Maîtrise d'Informatique
Session de septembre 2000

Techniques et Fondements de la Programmation (TFP)
partie "techniques"
le 11 septembre 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. (6p)

Un dictionnaire est un conteneur permettant d'insérer un objet (opération `insert`), supprimer un objet (opération `delete`), et de tester si un objet est dans le conteneur (opération `member`) selon une fonction de test d'égalité entre deux objets, donnée à la création du conteneur.

Écrire (en langage C) l'interface et l'implémentation d'un dictionnaire. Étant donné que vous n'avez pas de fonction d'hachage à votre disposition, les opérations doivent potentiellement parcourir tous les membres du conteneur. Il est donc raisonnable de l'implémenter sous la forme de liste simplement chaînée.

L'interface doit être impérative.

Exercice 2. (7p)

Les systèmes de fenêtrage comme X11 utilisent souvent une technique de tampons pour éviter des clignotements pendant la mise à jour d'une fenêtre. Pour cela, un programme client doit remplacer une suite d'opérations graphiques précédée par l'effacement de la fenêtre comme ceci :

- effacer la fenêtre `f`
- operation graphique 1 sur `f`
- ...
- operation graphique n sur `f`

par la suite suivante :

- créer un pixmap `p` de la taille de `f`
- operation graphique 1 sur `p`
- ...
- operation graphique n sur `p`
- copier `p` dans `f`
- détruire `p`

où un pixmap est une sorte de fenêtre non affichée.

Créer une macro Common Lisp `with-buffering` permettant de rendre la manipulation du pixmap invisible. Le programmeur doit donc pouvoir écrire :

```
(with-buffering *window*
  (graphics-operation-1 *window* ...)
  ...
  (graphics-operation-n *window* ...))
```

Pour cela, la macro doit générer du code pour :

- stocker la valeur de la variable donnée en paramètre à la macro dans une variable locale temporaire,
- créer un pixmap (on suppose l'existence d'une fonction `(create-pixmap window)`),
- stocker ce pixmap dans la variable donnée en paramètre à la macro,
- effectuer les opérations graphiques,
- copier et détruire le pixmap créé (on suppose l'existance des opérations `(copy-pixmap pixmap window)` et `(destroy-pixmap pixmap)`), et finalement
- rendre à la variable donnée en paramètre à la macro sa valeur initiale.

Attention, il est impératif que la valeur initiale de la variable donnée en paramètre à la macro soit préservée en cas d'erreur pendant les opérations graphiques.

Exercice 3. (7p)

Une file de priorités est un conteneur dans lequel à chaque élément est associée 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-p` renvoie une valeur indiquant si la file est vide.

En utilisant CLOS, écrire la classe de base et les déclarations des fonctions génériques pour modéliser la file de priorité. Cette classe sera une *classe protocole*; elle ne sera jamais instanciée. Les classes dérivées de la classe de base peuvent alors fournir plusieurs implémentations différentes.

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 (toujours en CLOS) une classe qui hérite de la classe de base pour l'implémentation de la file de priorité, ainsi que des méthodes d'implémentation pour les opérations de base.