Université BORDEAUX I

Année 1997-98
Session de janvier

Maîtrise d'informatique

Modèles et Techniques de Programmation (J.P.Braquelaire)

Durée: 2 heures
Tous documents autorisés

Première partie

1° **– 2 points –** On définit dans un langage de type LISP — il s'agit ici d'EMACS LISP — la fonction *++* au moyen de la définition :

```
(defun ++ (x) (setq x (+ 1 x)))
```

et l'on définit la variable numérique *n* initialisée à 1 par

```
(defvar n 1)
```

Comment s'effectue l'évaluation de la suite d'expressions :

```
(++ 3)
(++ n)
(++ n)
```

2° **– 2 points –** On définit maintenant la macro *++* au moyen de la définition

```
(defmacro ++ (x) `(setq ,x (+ 1 ,x)))
```

— on rappelle que l'opérateur « *accent grave* » (`) ou « *quasi-quotation* » est équivalent à l'opérateur « *accent aigu* », excepté pour les constructions précédées d'une virgule qui sont évaluées normalement. Comment s'effectue l'évaluation de la suite de macro-expressions

```
(++ 3)
(++ n)
(++ n)
```

3° **– 3 points –** On modifie la définition précédente en :

```
(defmacro ++ (x) `(setq ,x (+ increment ,x)))
(defvar increment 1)
```

Comment s'effectue l'évaluation, selon les différentes implémentations de LISP, de la suite d'expressions :

```
(let ((increment 5))
   (++ n))
(++ n)
```

Deuxième partie

La fonction *qsort* de la bibliothèque C standard

```
# include <stdlib.h>
void qsort(const void *début,
           size_t taille-vecteur, size_t taille-élément,
           int (*comparer)(const void *, const void *));
```

réalise le tri des ***taille-vecteur*** premiers éléments de taille ***taille-élément*** du vecteur ***début*** en utilisant la fonction de comparaison pointée par le pointeur de fonction ***comparer***. La fonction de comparaison reçoit l'adresse de deux objets à comparer et retourne une valeur négative, nulle ou positive, selon que le premier objet est respectivement inférieur, égal ou supérieur au second.

La fonction *bsearch* :

```
# include <stdlib.h>
void *bsearch(const void *clé, const void *début,
              size_t taille-vecteur, size_t taille-élément,
              int (*comparer)(const void *, const void *));
```

effectue la recherche d'un élément du vecteur trié ***début*** qui soit égal à l'élément d'adresse ***clé***. Les autres paramètres ont la même signification que pour la fonction *qsort*. S'il existe dans le vecteur ***début*** un élément égal à l'élément recherché pour la fonction de comparaison *comparer*, la fonction *bsearch* retourne son adresse sous la forme d'un pointeur générique. S'il en existe plusieurs la fonction *bsearch* peut retourner n'importe lequel d'entre eux. Elle retourne le pointeur *NULL* si aucun élément n'est trouvé.

4° **– 1 point –** En quoi peut-on dire que ces deux fonctions sont génériques?

5° **– 4 points –** On considère en C standard la classe *point* définissant des points dans le plan euclidien, et dont la méthode *point_compare* compare deux points en fonction de leur distance à l'origine. Illustrer par un exemple simple l'utilisation de la classe *point* pour définir un vecteur de points et de la fonction *qsort* pour trier ce vecteur.

6° **– 2 points –** Lorsqu'un vecteur contient plusieurs références à des valeurs égales modulo la fonction de comparaison, la fonction standard *bsearch* permet seulement d'obtenir une seule de ces références. On souhaite pouvoir les obtenir toutes, dans l'ordre dans lequel elles sont rangées dans le vecteur — cela nécessite évidemment une reprogrammation de la fonction *bsearch*. Proposer une façon d'effectuer cette extension en garantissant la compatibilité ascendante — on ne demande pas de donner la nouvelle implémentation de la fonction *bsearch*.

7° **– 4 points –** On considère la classe *array* codant un vecteur générique et contenant les deux méthodes *array_bsearch* et *array_qsort* dont les synopsys respectifs sont

```
# include <array.h>
void array_qsort(array self)
void *array_bsearch(array self, const void *clé)
```

Ces deux méthodes permettent de mettre en oeuvre les fonctions standard *qsort* et *bsearch* sur une instance de *array*. Donner l'interface de la classe et l'implémentation des méthodes *array_qsort* et *array_bsearch*.

8° **– 2 points –** Modifier l'exemple de la question 5 pour l'adapter à la classe *array*.