Université du Luxembourg 2004–2005

D.U.T. Informatique 2ème année mercredi 3 novembre

Programmation avancée en langage C++
Examen oral

`Sebastien.Varrette@imag.fr` (Bureau BR.4.07)

Durée: 30 minutes

Tout document interdit. Les exercices suivants sont à réaliser sur machine et devront être ensuite présentés et justifiés aux correcteurs.

Liste doublement chaînée

Une liste doublement chaînée est une liste avec un lien vers l'élément suivant mais aussi un lien vers l'élément précédent. La variable `start` pointe sur le premier élément de la liste et `start == NULL` si la liste est vide. De même, la variable `end` pointe sur le dernier élément de la liste, et `end == NULL` si la liste est vide. On dit que la liste est de tête `start` et de queue `end`.
Un exemple d'une telle liste est fourni en figure 1. Chaque élément de la liste

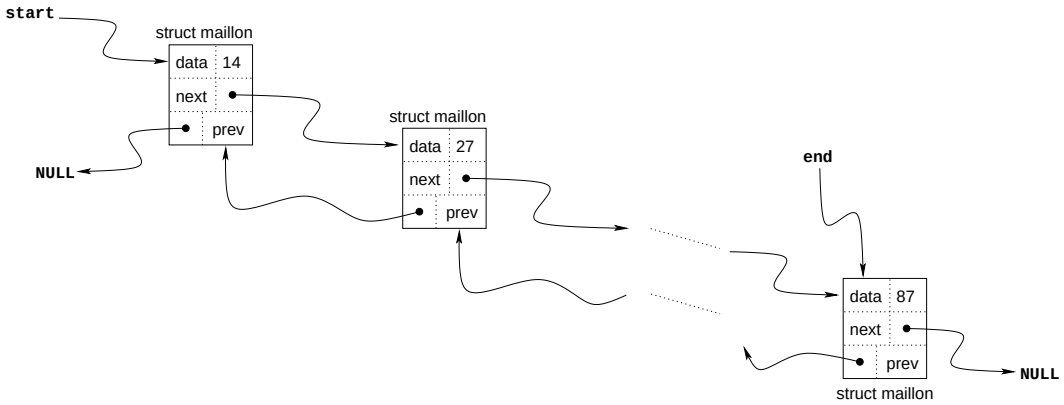

Figure 1 – structure de liste doublement chaînée ordonnée

est un *maillon* représenté par un triplet `<data,next,prev>`.

1. *Ecriture de structure*
   Définir la structure `Maillon` définissant les champs publiques `data`, `next` et `prev`.
2. *Ecriture de classe*
   Définir la classe `List` représentant une liste doublement chaînée d'entiers selon la représentation de la figure 1. Cette classe contiendra deux champs protégés :
   — un champ `_l` de type pointeur sur un `Maillon`
   — un champ `_nbElem` contenant le nombre déléments contenus dans la liste.
   En particulier, cette classe devra implémenter les méthodes publiques suivantes :

- un constructeur par défaut créant la liste vide ;
- un constructeur prenant en paramètre un entier n et initialisant une liste à 1 seul élément (n) ;
- une méthode `print()` affichant le contenu de la liste ;
- une méthode `getNbElem()` renvoyant le nombre d'éléments de la liste
- un destructeur (on veillera à libérer correctement la mémoire allouée pour **tous** les éléments de la liste. Pour cela, on pourra définir une fonction récursive `clear()` qui effectue la libération récursive de la mémoire et qui sera appelée dans le desctructeur)
- une méthode `addDataInHead(int d)` qui ajoute un nouveau maillon (de champ `data` initialisé à d) en tête de la liste ;

3. *Surcharge d'opérateurs* Surcharger l'opérateur `()` prenant en paramètre un indice `i` et permettant d'accéder directement (et eventuellement affecter) l'élément de la liste d'indice `i` (on vérifiera que $0 \leq i \leq$ `_nbElem`. Ainsi, pour la liste `l = (14, 27, 5)`, on doit pouvoir considérer la séquence d'instructions suivantes :

```
l.print();             // affiche 14 -> 27 -> 5 -> NULL
cout << l(0) << endl; // affiche 14
l(0) = 2;              // Affectation
l.print();             // Affiche 2 -> 27 -> 5 -> NULL
```

4. *Héritage*

   Définir la classe `PriorityList` qui hérite de la classe `List` en ajoutant un paramètre entier caractérisant une valeur de priorité de la liste. Illustrer l'utilisation de cette classe en faisant intervenir les différentes méthodes qui la compose.

5. *Utilisation de modèle*

   Reprendre la définition de la structure `Maillon` et de la classe `List` en utilisant cette fois-ci les `template` pour permettre de gérer un champ `data` de n'importe quel type. On définira ainsi la structure `TMaillon` et la classe `TList` et on illustrera son utilisation sur une liste de caractères et de flottants.