TD2 : Implémentation des listes

Vendredi 4 février 2011

Ce TD vise à mettre en place une bibliothèque minimale pour la manipulation des listes chaînées, qui servira au TD suivant. Les questions portent essentiellement sur l'implémentation de diverses opérations. Pour chacune de ces opérations, il vous est demandé :

- de proposer un exemple d'utilisation de l'opération pour tester votre implémentation,
- dans le cas où votre opération crée ou modifie une liste, de représenter l'état de la mémoire avant et après appel à votre fonction,
- de justifier que votre implémentation termine et produit un résultat correct,
- d'évaluer la complexité de votre algorithme.

Remarque : vous pouvez choisir le style (récursif ou itératif) qui vous convient le mieux. Il est néanmoins fortement conseillé de faire chaque question dans les deux styles (quand cela s'y prête).

Représentation mémoire des listes

1. Proposer un type C pour représenter les listes d'entiers.
2. Écrire un constructeur de liste vide, et un constructeur `cons` qui ajoute un élément en tête de liste et de prototype

   ```
   list cons(int h, list t);
   ```

3. Écrire une fonction `free_list` permettant de libérer la mémoire allouée à une liste.
4. Écrire une fonction de copie de liste, de prototype

   ```
   list copy_list(list l);
   ```

5. Implémenter une fonction produisant la liste des n premiers entiers naturels dans l'ordre décroissant, de prototype

   ```
   list list_integers(int n);
   ```

   Si on l'appelle avec 5 pour argument la fonction doit produire la liste $4 :: 3 :: 2 :: 1 :: 0 :: \emptyset$.
   - (Optionnel) Comment produire la même liste en ordre croissant ?
6. (Optionnel) De la même façon que l'on peut utiliser des pointeurs sur des données (comme des entiers ou des caractères), on peut également manipuler en C des **pointeurs sur fonction**. Lorsque l'on définit une fonction, son nom peut être utilisé pour désigner l'adresse en mémoire du code de la fonction : le nom de la fonction est un pointeur sur la mémoire qui contient le code. Cette possibilité a un intérêt majeur, celui de permettre le passage d'une fonction en paramètre d'une autre. Réciproquement, si `pf` est un pointeur sur fonction alors on peut appeler la fonction pointée par `pf` en utilisant l'opérateur d'indirection : `(*pf)(...)`. Sachant cela, proposer une fonction `list_init` qui initialise une liste en appelant une fonction sur les entiers données en paramètres. Par exemple, l'appel à `list_init(3,pf)` doit retourner la liste $f(2) :: f(1) :: f(0) :: \emptyset$ où f est la fonction pointée par `pf`. Le prototype attendu est :

   ```
   list list_init(int n, int (*pf)(int x));
   ```

Observateurs

Implémenter successivement :

1. la fonction `length` qui calcule la longueur d'une liste,
2. la fonction `member` qui teste la présence d'un élément dans une liste, puis la fonction `multiplicity` qui calcule le nombre d'occurrences d'un élément dans une liste. Proposer une autre version de `member` en utilisant `multiplicity`,
3. la fonction `print_list` qui affiche une liste à l'écran,
4. la fonction `equals` qui teste l'égalité de deux listes,
5. la fonction `get` qui retourne le i^e élément d'une liste.

Modification des listes

1. On s'intéresse à l'opération qui permet de supprimer le premier élément d'une liste. À quoi doit-on prendre garde ? Quelle est la conséquence sur le prototype et sur l'usage de cette fonction ?
2. Implémenter l'opération `chop` qui supprime le premier élément d'une liste.
3. Utiliser la réponse précédente pour la fonction `remove` qui retire le i^e élément d'une liste.
4. Écrire la fonction `insert` qui permet d'insérer un élément à la i^e position d'une liste, ainsi qu'une variante qui insère un élément à la fin de la liste.
5. Comment doit-on s'y prendre pour insérer un élément au milieu d'un tableau ? Discuter l'efficacité de l'opération "insertion d'un élément" dans le cas d'un tableau et d'une liste.