TD/TP 9 : utilisation de listes

Gilles Peskine

6 décembre 2002

1 Opérations de base

Voici (un morceau de) l'en-tête d'une bibliothèque de manipulation de listes. Pour l'instant, les listes seront stockées sous la forme d'un tableau, dont on fera varier la taille.

```
typedef struct boite_liste {
    int longueur;
    int *tab;
} *liste;

liste liste_vide(void);
/* Fabrique une nouvelle liste, vide. */
int est_vide(liste);
/* renvoie vrai ou faux : la liste est-elle vide ? */
int valeur_element(liste, int);
/* (liste, position)
   renvoie l'élément stocké dans la liste à la position donnée. */
void ajoute_fin(liste, int);
/* (liste, element).
   Ajoute un nouvel élément à la fin de la liste. */
void insere(liste, int, int);
/* (liste, position, element)
   Insère un nouvel élément dans la liste, à la position donnée.
   Par exemple insere(x,1,1) insère x juste après le premier élément. */
void supprime_element(liste, int);
/* (liste, position)
   Retire de la liste l'élément à la position donnée. */
```

Écrire les fonctions mentionnées ici.

2 Recopie

Écrire une fonction qui recopie une liste.

```
liste copie_liste(liste);
```

3 Itérations

De manière générale, on appelle *itérateur* une fonction qui applique une fonction à tous les éléments d'une liste.

Définissons un type pour les fonctions transformant un entier en un entier :
`typedef int transformation_entiere(int);`
ainsi que pour les fonctions qui agissent sur un entier :
`typedef void procedure_entiere(int);`

La fonction `iter_liste` applique successivement `f` à tous les éléments d'une liste :
`void iter_liste(procedure_entiere *f, liste l)`

La fonction `map_liste` transforme une liste en remplaçant successivement chaque élément de `f` par son image par `f`.
`void iter_liste(transformation_entiere *f, liste l)`

Écrire les fonctions `iter_liste` et `map_liste`.

Utiliser `iter_liste` pour écrire une fonction d'affichage d'une liste.

4 Quelques fonctions usuelles

(a) Écrire une fonction qui recerche un élément dans une liste. Renvoyer 1 si l'élément est présent, et 0 sinon.
`int recherche(liste l, int x);`

(b) Écrire une fonction qui concatène deux listes (créer une nouvelle liste pour stocker le résultat).
`liste concatene(liste l1, liste l2);`

(c) Écrire une fonction qui renverse une liste (inverse l'ordre des éléments). Modifier la liste sur place.
`void renverse(liste l);`

5 Tri

(a) Écrire une fonction qui teste si une liste est triée en ordre croissant.

(b) Écrire une fonction qui prend comme arguments un nombre entier et une liste triée en ordre croissant, et qui insère le nombre à la bonne position dans la liste.
`void insere_croissant(liste l, int x);`

(c) Utiliser la fonction précédente pour écrire une fonction qui fait une copie triée d'une liste quelconque. `liste trie_liste(liste l);`