TD/TP 11 : listes

Gilles Peskine

20 décembre 2002

1 Structure de données

Définition : une liste d'entiers est soit vide, soit une paire constituée d'un entier et d'une liste d'entiers.

Voici plusieurs tentatives de définition du type des listes. Pour chacune, indiquer si elle est correcte ou non. Comparer les définitions correctes. Comment définir une liste vide ?

```
typedef struct {
    int x;
    liste1 q;
} liste1;

struct li2 {
    int x;
    liste2 q;
};
typedef struct li2 liste2;

struct li3 {
    int x;
    liste3 q;
};
typedef struct li3 *liste3;
```

```
struct li4 {
    int x;
    struct li4 q;
};
typedef struct li4 *liste4;

struct li5 {
    int x;
    struct li5 *q;
};
typedef struct li5 liste5;

struct li6 {
    int x;
    struct li6 *q;
};
typedef struct li6 *liste6;
```

```
struct li7 {
    int x;
    struct li7 *q;
};
typedef struct li7 **liste7;

struct li8;
typedef struct li liste8;
struct li8 {
    int x;
    liste8 q;
};

struct li9;
typedef struct li *liste9;
struct li9 {
    int x;
    liste9 q;
};
```

Nous utiliserons la définition 6.

Terminologie : une liste d'entiers non vide ℓ est une paire constituée d'un entier x et d'une liste d'entiers q : l'entier x est appelé *tête* de la liste, et la liste q est appelée *queue* de la liste ℓ.

Écrire les fonctions suivantes :

```
int est_vide(liste l);
int tete(liste l);
liste queue(liste l);
liste cons(int x, liste l);
```

La fonction `cons` (un nom traditionnel, qui vient de « construire ») fabrique et renvoie une liste dont la tête est `x` et la queue est `l`.

2 Parcours de base

(a) Écrire une fonction qui détermine la longueur d'une liste.

(b) Écrire une fonction qui renvoie le i-ème élément d'une liste.

```
size_t longueur(liste l); int element(liste l, size_t i);
```

3 Entrées-sorties

(À faire en TP.)

(a) Écrire une fonction qui lit une liste d'entiers depuis `stdin`. Comment peut-on indiquer la fin de la liste ?

(b) Écrire une fonction d'affichage d'une liste d'entiers.

4 Ajout

(a) Écrire une fonction qui insère un entier dans une liste à une position donnée. Par exemple, `insere(&l,0,42)` insère 42 au début de la liste, tandis que `insere(&l,1,42)` insère 42 après le premier élément.

```
void insere(liste *l, size_t i, int x);
```

Pourquoi un argument de type `liste*` (c'est-à-dire `struct li **`) et non `liste` ?

(b) Écrire une fonction qui ajoute un entier à la fin d'une liste.

```
void ajoute(liste *l, int x);
```

(c) La fonction `ajoute` prend un temps proportionnel à la longueur de la liste. Comment modifier la structure de données retenue pour les listes, si on veut rendre l'opération d'ajout plus efficace ?

5 Suppression

(a) Écrire une fonction `supprime` qui retire le `i`-ème élément d'une liste.

(b) Écrire une fonction `supprime_dernier` qui retire le dernier élément d'une liste.

6 Quelques fonctions classiques

(a) Écrire une fonction qui concatène deux listes (les met bout à bout).

(b) Écrire une fonction qui recopie une liste.

(c) Écrire une fonction qui retourne une liste. Par exemple, la liste $[3, 42, 18]$ doit être transformée en $[18, 42, 3]$.

(d) Écrire une fonction qui fait une copie d'une liste en la retournant.

(e) Écrire une fonction qui compare deux listes. (Deux listes sont égales si et seulement si elles ont les mêmes éléments aux mêmes positions.)

Note : une fonction qui transforme une liste en modifiant la liste existante, comme la fonction de retournement du *(c)*, est dite *destructrice*. Une fonction qui fabrique une nouvelle liste en ne faisant que lire l'ancienne est dite *non destructrice*.

7 Fonctionnelles

(a) Écrire une fonction `iter` qui applique une fonction à tous les éléments d'une liste. Par exemple, si `l` a trois éléments 3, 42 et 18, alors `iter(l, f);` équivaut à

```
f(3); f(42); f(18);
```

(b) Écrire une fonction `map` qui à partir d'une fonction f et d'une liste $[x_0, x_1, x_2, \ldots]$ fabrique la liste $[f(x_0), f(x_1), f(x_2), \ldots]$.

(c) Écrire une fonction `extrait` qui fait une copie d'une liste, mais en ne conservant que les éléments qui vérifient une certaine propriété. Par exemple :

```
int positif(int x) { return (x > 0); }
```

Si `l` est la liste $[7, -1, 42, -3, -1, 8, 9]$ alors `extrait(l, &positif)` donne la liste $[7, 42, 8, 9]$.

8 Tris

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

(b) Écrire une fonction qui prend comme arguments une liste d'entiers, qu'on suppose triée, et un entier, et qui insère l'entier dans la liste au bon endroit pour que la liste résultante soit triée.

(c) En déduire une méthode pour trier une liste : le tri par insertion.

(d) Écrire une fonction qui fusionne (destructivement de préférence) deux listes triées. Par exemple, la fusion de $[3, 8, 9]$ avec $[0, 5, 5, 8, 8]$ est une liste $[0, 3, 5, 5, 8, 8, 8, 9]$.

(e) Écrire une fonction qui coupe une liste en deux au milieu.

(f) Utiliser les fonctions du *(d)* et du *(e)* pour faire une fonction qui trie une liste. (Note : le code n'est pas particulièrement complexe, mais il y a une difficulté conceptuelle.)