Initiation à la programmation en C

TP n°10

Antoine Miné

10 mai 2007

Site du cours: `http://www.di.ens.fr/~mine/enseignement/prog2006/`

Exercice 1. Génération de listes.
On rappelle le type suivant pour les cellules d'une liste simplement chaînée d'entiers :

```
struct cell {
  struct cell* next;
  int          data;
};
```

Programmez une fonction :

```
struct cell* liste_aleatoire(int nb);
```

qui crée une liste de `nb` éléments aléatoires (voir la fonction `lrand48`) et renvoie la tête de la liste créée. On pourra procéder par insertions successives en tête de liste.
Programmez également une fonction :

```
void affiche_liste(struct cell* head);
```

qui affiche à l'écran la contenu de la liste.

Exercice 2. Retournement de listes.
Écrivez une fonction :

```
struct cell* retourne_liste_copie(struct cell* head);
```

qui crée une copie d'une liste où les éléments apparaissent dans l'ordre inverse. La liste originale n'est pas modifiée.
Écrivez ensuite une fonction :

```
struct cell* retourne_liste_en_place(struct cell* head);
```

qui retourne la liste "en place", c'est à dire, sans en faire de copie. À la place, les pointeurs `next` de la liste passée en argument seront modifiés.
Testez vos fonctions grâce aux fonctions de génération de listes aléatoires et d'affichage du premier exercice. Vérifiez en particulier la correction aux cas limites (liste vide, liste à un élément) et l'absence d'erreur ou de fuite de mémoire (testez pour cela votre programme avec `valgrind`).

Exercice 3. Tri par fusion.
Le tri par fusion est un algorithme de tri récursif particulièrement adapté aux listes. Il fonctionne de la manière suivante :
– si la liste contient plus d'un élément :
  – on découpe la liste en deux listes de taille moitié,
  – chacune de ces listes est triée séparément, par un appel récursif,
  – les deux listes triées sont fusionnées en une unique liste triée ;
– si la liste est vide ou contient un élément, il n'y a rien à faire.
On utilisera ici des listes simplement chaînées d'entiers avec un type de cellule identique à celui des exercices précédents.
Programmez d'abord une fonction :

```
struct cell* decoupe(struct cell* head);
```

qui découpe la liste pointée par `head` en deux listes de taille moitié. En sortie de fonction, le pointeur de tête original `head` pointera sur la tête d'une des deux listes tandis que la tête de l'autre liste sera retournée. `decoupe` ne doit pas allouer de nouvelle cellule ou toucher aux champs `data` mais uniquement modifier les pointeurs `next`.
Programmez ensuite une fonction :

```
struct cell* fusion(struct cell* head1, struct cell* head2);
```

qui fusionne les deux listes triées `head1` et `head2` en une unique liste, toujours triée, et renvoie la tête de cette liste. Comme `decoupe`, cette fonction se contente de réorganiser les pointeurs `next` des cellules existantes.
Déduisez-en une fonction récursive de tri par fusion :

```
struct cell* tri(struct cell* head);
```

Testez votre tri grâce aux fonctions de génération de listes aléatoires et d'affichage du premier exercice.

Exercice 4. Tri générique.
Proposez un type de cellule correspondant à des listes chaînées de chaînes de caractères. Modifiez ensuite la fonction de tri par fusion pour qu'elle prenne en argument un pointeur sur une fonction de comparaison de type `int (*compare)(char*,char*)`. Le tri utilisera cette fonction à chaque fois qu'il doit comparer deux chaînes de caractères ; on supposera que la fonction renvoie 0 si les deux chaînes sont égales, un entier strictement positif si la première chaîne est "plus grande" que la seconde et un entier strictement négatif sinon.
Testez votre fonction de tri avec les fonctions de comparaison suivantes :
– la fonction standard `strcmp` qui compare deux chaînes pour l'ordre lexicographique (*i.e.*, du dictionnaire),
– la fonction standard `strcasecmp` similaire à `strcmp`, mais insensible à la casse,
– une version `monstrcmp` de `strcmp` gèrant correctement les accents, à programmer par vos soins.