Université Lille1
Sciences et Technologies

Licence d'informatique
Module de Pratique du C

Travaux pratiques

Ma commande (quick)sort

Novembre 2005
mise à jour d'Octobre 2010

L'objet du TP est d'écrire une commande de tri équivalent de la commande Unix `sort` basée sur l'algorithme de tri quicksort.

1 Le tri quicksort

Le tri quicksort a été introduit par C.A.R. Hoare en 1960. C'est le prototype des méthodes *diviser pour résoudre*.

On suppose disposer :
- d'un tableau composé d'éléments à trier ;
- d'une fonction f de comparaison de deux éléments du tableau :
  - f retourne 0 si les deux éléments sont équivalents ;
  - f retourne une valeur positive si le premier élément est considéré plus grand que le second ;
  - f retourne une valeur négative sinon.

Principe du tri quicksort. Le principe du tri quicksort est le suivant : pour trier un tableau, on considère x, le premier élément du tableau. Supposons qu'après manipulation, on réussi à réagencer le tableau en deux zones contiguës :
- la première zone constituée des éléments$y$ tels que$f(y, x) < 0$;
- la seconde zone constituée des éléments z tels que $f(x, z) \leq 0$.

Il suffit de recommencer récursivement la manipulation sur les première et seconde zones du tableau pour trier notre tableau. La récursion prend fin lorsque une zone du tableau est réduite à un élément.

Attention. Aucune copie du tableau n'est faite lors du tri : ce tri est *destructif*, il altère le tableau à trier.

Une méthode de partitionnement. Une implantation du tri quicksort réside sur un algorithme de partitionnement d'un tableau en deux zones. Pour partitionner un tableau, on peut opérer comme suit :
- on choisit comme valeur pivot x, la valeur du premier élément comme élément pivot x (on peut aussi choisir n'importe quel autre valeur, par exemple la valeur de l'élément médiant des 3 ou 5 premiers éléments du tableau) ;
- initialement, on considère que `montant` et `descendant` référencent respectivement le premier et dernier éléments du tableau ;
- tant que `montant` est inférieur à decendant, on itère les phases suivantes :
  - incrémenter `montant` tant que l'élément y référencé est tel que $f(y, x) < 0$;
  - décrémenter `descendant` tant que l'élément z référencé est tel que $f(x, z) \leq 0$;
  - si `montant` et `descendant` se rencontrent, quitter ;
  - échanger les éléments référencés par `montant` et `descendant`.

  Pour finir, on échange le pivot et `descendant`.

2 Tri quicksort d'un tableau d'entiers

Dans un premier temps nous allons appliquer le principe du tri quicksort à un tableau d'entiers.

Initialisation aléatoire d'un tableau. On utilisera la fonction de la bibliothèque standard :

```
#include <stdlib.h>
int rand (void);
```

qui retourne entier pseudo-aléatoire entre 0 et `RAND_MAX` (voir le manuel en ligne) pour initialiser un tableau de taille

```
#define TABSIZE 1000
```

Fonction de tri d'un tableau d'entiers. Implanter suivant le principe du tri quicksort la fonction de prototype :

```
void quicksort_int(int tab[], unsigned int nelem);
```

qui trie le tableau `tab` de `nelem` entiers.

Tester cette fonction sur un tableau aléatoirement rempli.

3 Quicksort générique

On généralise le cas du tri d'un tableau d'entiers pour trier des tableaux de type quelconque. Il s'agit de proposer votre propre implémentation de la fonction `qsort()` de la bibliothèque standard (`man 3 qsort`).

Écrivez le code source de la fonction de prototype suivant :

```
void quicksort(void *base, int nelem, int size,
               int(*compar)(const void *, const void *));
```

avec

- `base` une référence sur le premier élément du tableau à trier ;
- `nelem` le nombre d'éléments du tableau à trier ;
- `size` la taille, en octets, d'un élément du tableau ;
- `compar` un pointeur sur la fonction de comparaison. Cette fonction accepte deux références sur les éléments à comparer et retourne la valeur de la comparaison selon le principe de la fonction .

4 Commande de tri

Pour terminer, il s'agit de produire une commande `msort`, version limitée de la commande Unix `sort`.

La commande `msort` trie les lignes fournies sur l'entrée standard et produit le résultat sur la sortie standard. Elle ne prend aucun argument sur la ligne de commande.

Votre implantation pourra être limitée à des tris

- de fichiers comportant au plus `NMAXLINE` lignes ;
- chaque lignes comportant au plus `NMAXCHAR` caractères.

Quelle structure de données allez vous utiliser pour mémoriser l'ensemble des lignes d'un fichier ?

Fournissez des fonctions permettant de nourrir une telle structure de données depuis les lignes lues sur l'entrée standard et permettant d'afficher sur la sortie standard une telle structure.

Vous utiliserez la fonction `strcmp()` de la bibliothèque standard comme fonction de comparaison de lignes.

```
#include <string.h>
int strcmp (const char *s1, const char *s2);
```

Vous êtes invités à réutiliser le code que vous aviez écrit pour la fonction `readl()` lors du TP précédent.

5 Travail à rendre

Vous devez rendre :

1. Une bibliothèque de tri de tableau d'entiers (fichiers `qsint.c` et `qsint.h`).
2. Un programme de test de cette bibliothèque (`qsint_tst`).
3. Une bibliothèque de tri quicksort générique (fichiers `qs.c` et `qs.h`).
4. Votre implantation de la commande `msort`.