TD2: Pointeurs, structures, listes chaînées

Octobre 2014

Exercice 1. Pointeurs

a. Afficher (et comprendre) la taille en mémoire des types suivants :

```
char; int; double; char*; void*; int*; double*; int**; int[10]; char[7][3]; int[]
```

Soit `tab` un tableau de 10 caractères déclaré par `char tab[10];`
Afficher la taille mémoire de

```
tab; tab[0]; &tab[0]; *&tab; *&tab[0]
```

Soit `p` une variable définie par `char (*p)[10] = &tab`
Afficher la taille mémoire de

```
p; *p; (*p)[2]; &(*p)[2];
```

b. Ecrire une fonction qui ne renvoie rien et dont l'effet de bord est de permuter la valeur de deux entiers passés en paramètres.

c. Que produit l'appel à la fonction suivante ?

```
void reinitPointeur(int* p){
   p = NULL;
}
```

Pour tester, on peut par exemple exécuter les instructions suivantes dans la fonction `main()` en ajoutant les instructions d'affichages adéquates :

```
int a = 1;
int* p = &a;
reinitPointeur(p);
```

Comment modifier la fonction pour que la réinitialisation du pointeur soit effective ?

Exercice 2. Structure, tableaux

a. Définir une structure appelée `Tableau` qui contient deux champs :

1. `taille` : un entier.
2. `tab` : un tableau de taille 100 dont les `taille` premiers entiers sont significatifs ;

b. Quelle est la taille mémoire de cette structure ?

c. Ecrire une fonction

```
int alea(int n)
```

qui renvoie un entier tiré aléatoirement entre `0` et `n`. Pour cela voir la documentation des fonctions `rand()` et `srand()` de la librairie `stdlib`.

d. Ecrire une fonction qui initialise une variable de la structure : `taille` vaut 10, et le tableau est rempli avec 10 entiers choisis aléatoirement entre 0 et 20.

e. Ecrire une fonction qui affiche les éléments du tableau.

f. Ecrire une fonction qui retourne le produit des éléments du tableau.

g. Ecrire une fonction qui retourne la valeur minimale du tableau.

h. Ecrire une fonction qui effectue un décalage de 1 case à droite de tous les éléments d'un tableau. La valeur 0 est affectée à la case à gauche. La taille du tableau est donc augmentée de 1.

i. Ecrire une fonction qui trie les éléments du tableau.

j. Ecrire une fonction qui insère une valeur dans un tableau trié. Après l'insertion, le tableau est toujours trié. La taille du tableau est augmentée de 1.

k. Ecrire une fonction qui inverse les éléments d'un tableau. Cette inversion s'effectue sur le tableau lui-même sans utiliser de tableau intermédiaire.

l. Ecrire une fonction qui supprime un élément choisi au hasard dans le tableau. La taille du tableau est diminuée de 1.

m. Ecrire une fonction qui élimine les valeurs en double (ou plus) du tableau.

n. Programmer le tri par selection, le tri par insertion, le tri à bulle.

o. Instrumenter le code pour que les fonctions de tri renvoie le nombre d'accès en lecture et écriture du tableau.

p. Pour une taille de tableau fixée, générer plusieurs tableaux et calculer le nombre moyen de comparaisons effctuées.

q. Faire varier la taille des tableaux (par exemple de 100 à 1500 par pas de 100), et afficher à chaque fois le nombre moyen de comparaisons effectuées par les différents méthodes de tri. Les valeurs obtenues peuvent être affichées sur une courbe de la manière suivante :

1. Faire en sorte que l'affichage soit sur 2 colonnes où
   — la première colonne contient la taille des tableaux générés
   — la deuxième colonne contient le nombre moyen de comparaisons
2. Rediriger les affichages dans le terminal vers un fichier. Si l'exécutable s'appelle `a.out`, cela se fait avec la commande

   ```
   ./a.out > resultat
   ```

3. Démarrer le logiciel `gnuplot` dans le terminal.
4. Afficher les résultats avec la commande

   ```
   gnuplot> plot "./resultat" with lines notitle
   ```

5. Pour comparer la courbe obtenue avec le graphe d'une fonction `f` (ici le nombre de comparaison "théorique") :

   ```
   gnuplot> replot f(x)
   ```

Cette méthode pourra être utilisée par la suite pour visualiser le nombre d'opérations moyen effectuées par les tris.

r. Implementer un algorithme de tri en temps linéaire quand les valeurs des tableaux sont restreintes à l'intervalle $[0 \dots 100]$.

s. Implémenter le tri rapide.

Exercice 3. Listes chaînées

Dans cette section, on programmera les fonctions de manipulation de listes suivantes en utilisant des fonctions récursives lorsque cela est possible :

a. Définir la structure de données `Liste` qui implémentera la liste chaînée.

b. Les fonctions de base : créer une liste vide, tester si une liste est vide, afficher une liste, libérer la mémoire.

c. Fonctions d'ajout : ajout au début, ajout à la fin, ajout trié. Pour la dernière insertion, on écrira au préalable une fonction qui vérifie que la liste est bien triée. Pour tester ces fonctions, on créera une liste en ajoutant des entiers générés aléatoirement entre 0 et n jusqu'à ce que 0 soit tiré.

d. Retourner le nombre d'éléments de la liste.

e. Recherche un élément.

f. Supprimer un élément.

g. Concaténer deux listes.

h. Entrelacer deux listes triées pour former une nouvelle liste triée.

i. Implémenter le tri à bulles sur une liste.

j. Renverser une liste chaînée.

k. Implémenter le tri fusion dans une liste d'entiers.

l. Implémenter le crible d'Eratosthene avec une liste afin de trouver tous les nombres premiers plus petits qu'une valeur passée en argument du programme.

m. Étant donnée une liste d'entiers, on cherche à les classer selon leur valeur modulo un entier K passé en paramètre de la fonction que vous écrirez. Par exemple, si $K = 2$, on veut séparer les valeurs paires des valeurs impaires. Proposer une structure adaptée au classement et implémenter le classement.