Programmation avancée
Examen du 20 mai 2015

Exercice 1 : représentation d'une matrice

Donner plusieurs façons de représenter une matrice (un tableau à deux dimensions de `int`) en mémoire.

Exercice 2 : mémoire

On considère le code suivant. Donner une représentation de la mémoire (en distinguant bien les différents segments, et en donnant le nom et le contenu éventuels des données) lorsque chaque appel de la fonction `f` commence, ainsi que juste avant chaque `return` de `f`.

```
int f(int x) {
   static int y = 10;
   int *p = (int *) malloc(3*sizeof(int));
   p[0] = 5 * y;
   y++;
   int r = p[0]+1;
   if(x>1)
      r = r + f(x-1);
   return r;
}
int main() {
   printf("%i\n", f(3));
   return 0;
}
```

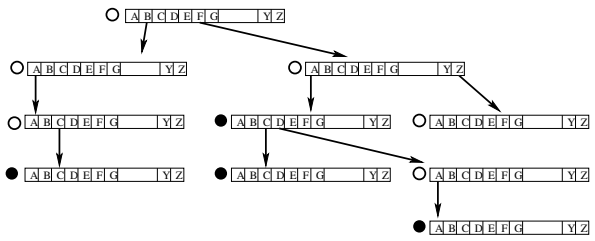
Problème : stockage sur disque d'un dictionnaire

Dans ce problème, un **dictionnaire** est un ensemble de mots sur l'alphabet $\{A, ...Z\}$ (lettres majuscules, sans ponctuation ni rien d'autre). On travaillera sur deux stockages du dictionnaire :

Un **fichier-dictionnaire** est un fichier texte, sur disque. Il est constitué des mots du dictionnaire *triés dans l'ordre alphabétique*, chaque mot (même le dernier) étant terminé par un `'\n'` (il y a donc un mot par ligne). Exemple :

```
BAC
FA
FAC
FADA
```

Un **arbre-dictionnaire** est un arbre *en mémoire*. Chaque nœud a 26 pointeurs sur des nœuds fils, plus un entier disant si les lettres lues sur la branche allant du nœud-racine à ce nœud forment un mot du dictionnaire (l'entier est alors à 1, et à 0 sinon). Le même dictionnaire peut être stocké ainsi (un rond noir est un 1 et un rond blanc un zero) :



Les exercices du problème sont indépendants : on peut donc faire chaque exercice en utilisant, sans les avoir écrites, les fonctions des autres exercices. Vous pouvez créer des fonction auxilliaires.

Exercice 3 : comptage des lignes

Écrire une fonction `int nbLignes(FILE *fich)` qui compte le nombre d'occurrences du caractère `'\n'` dans le fichier `fich`. On supposera sans le vérifier que `fich` a été correctement ouvert en lecture.

Exercice 4 : taille d'une ligne

Écrire une fonction `int tailleLigne(FILE *fich)` qui compte le nombre de caractères dans le fichier `fich` situés entre le pointeur de fichier courant (la tête de lecture virtuelle) et le prochain caractère `'\n'` de ce fichier. Par exemple, si on est au tout début d'un fichier dont la première ligne est `BAC`, la fonction doit retourner 3. La fonction doit retourner 0 si on se situe sur un `'\n'`, et -1 en cas d'erreur. Le pointeur de fichier courant doit être situé au même endroit à la fin de l'appel qu'au début. On supposera sans le vérifier que `fich` a été correctement ouvert en lecture

Exercice 5 : lecture d'un fichier

Écrire une fonction `char **mots(char *nomDeFichier)` qui, étant donné le nom d'un fichier-dictionnaire, si ce fichier contient n mots, alloue et renvoie un tableau de n chaînes, la $i$ème chaîne étant le $i$ème mot du dictionnaire.

Dans notre exemple, la fonction va donc allouer un tableau de quatre chaînes, la première étant `"BAC"`.

La fonction doit retourner `NULL` en cas d'erreur, par exemple si le fichier ne peut pas être ouvert ou n'est pas au format d'un fichier-dictionnaire.

Exercice 6 : type d'un nœud

Définir un nœud de l'arbre-dictionnaire, et le type `pnoeud` d'un pointeur sur un nœud de l'arbre-dictionnaire. Si vous n'y arrivez pas, ce type pourra s'appeller `struct noeud *` (et dans tous les exercices suivants vous pourrez remplacer `"pnoeud"` par `"struct noeud *"`). En plus des autres champs, chaque nœud doit avoir un pointeur vers son père (qui vaudra `NULL` pour la racine).

Exercice 7 : insertion dans le dictionnaire

Écrire une fonction `int inserer(pnoeud R, char *mot)` qui insère un mot dans le dictionnaire de racine `R`. Elle renvoie -1 si erreur, 0 si OK.

Exercice 8 : lecture d'un fichier-dictionnaire

Écrire une fonction `pnoeud litArbreDic(char *nomDeFichier)` qui, étant donné le nom d'un fichier-dictionnaire, construit un arbre-dictionnaire de ce dictionnaire et renvoie un pointeur sur sa racine. En cas d'erreur `NULL` sera renvoyé. Pensez à libérer la mémoire qui sera inutilisée par la suite.

Exercice 9 : profondeur d'un nœud

Écrire une fonction `int prof(pnoeud N)` donnant la profondeur d'un nœud, c'est-à-dire sa distance à la racine (*NB :* la racine a donc profondeur 0).

Exercice 10 : mot correspondant à un nœud

Écrire une fonction `char *mot(pnoeud N)` qui, étant donné un nœud `N` d'un arbre-dictionnaire, renvoie le mot qu'il faut lire depuis la racine pour arriver à ce nœud. Dans l'arbre de l'exemple, `mot` appellée sur le nœud le plus en bas à droite allouera et renverra la chaîne `"FADA"`.

Exercice 11 : nettoyage d'un arbre-dictionnaire

Vous avez peut-être remarqué que l'arbre-dictionnaire de l'exemple contient un nœud inutile (celui correspodant au chemin FZ). Écrire une fonction `void nettoieArbre(pnoeud R)` qui, étant donné un arbre-dictionnaire de racine `R`, supprime (et libère) tous les nœuds inutiles, c'est-à-dire ceux qui ne sont pas sur le chemin de la lecture d'un mot du dictionnaire.

Exercice 12 : écriture d'un arbre-dictionnaire

Écrire une fonction `int ecritArbreDic(char *nomDeFichier, pnoeud R)` qui, étant donné un arbre-dictionnaire de racine `R`, écrit (en écrasant un contenu préexistant au besoin) un fichier-dictionnaire `nomDeFichier` contenant son dictionnaire. La fonction renverra -1 en cas d'erreur, 0 sinon.

Vous n'êtes pas obligé d'utiliser les fonctions des exercices précédents, mais attention, on ne peut pas supposer que les mots ont une taille maximum connue à la compilation. Pensez à libérer la mémoire qui sera inutilisée par la suite.