TP 3 Algorithmique Avancée : Représentation d'Arbres

Serigne A. Gueye

CERI - Licence 2 Informatique

Nous souhaitons coder une classe *Arbre* permettant de simuler la gestion des répertoires d'un système d'exploitation de type *Unix* (ou *Linux*). Nous considérons à la racine le répertoire "/" (root) dans lequel se trouve hiérarchiquement tous les autres. Cette hiérarchie sera représentée sous la forme d'un arbre.

Comme le nombre de sous-répertoires d'un répertoire donné n'est pas limité à priori, nous optons pour une représentation sous forme d'arbres à **ramifications (nombre de fils) non bornées (voir cours)**.

1/ Ecrire la structure de données (*noeud*) représentant un noeud de l'arbre. Elle comportera, en plus des pointeurs liés aux ramifications (*pere*, $fils\ gauche$, $frere\ droit$) un champ de type $char*$ (**"***nom***"**) contenant le nom du répertoire.

2/ Créer la classe suivante :

- **C++**
```
class Arbre
{
        noeud * root ;
        noeud * current ;

        public :
        Arbre() ;
        ~Arbre() ;
}
```

où "*root*" est la racine (i.e celle du repertoire "/"), et "*current*" celle du répertoire dit **courant**. C'est à dire celui dans lequel l'on se trouvera en parcourant les répertoires.

Le constructeur créera un arbre comportant un noeud de nom "/" ayant deux fils nommés : "." et "..".

Le fils gauche de "." aura la même adresse que son père, et le fils gauche de ".." la même adresse que son grand-père.

De façon générale "." et ".." devront figurer **automatiquement** dans tout répertoire créé, et avec les mêmes caractéristiques qu'expliquées ci-dessus.

root et *current* pointent initialement sur "/".

Le destructeur détruira entièrement l'arbre par le biais d'une méthode récursive à écrire :

$$Suppression(noeud * r).$$

3/ Ajouter la méthode

$$pwd()$$

renvoyant le nom du répertoire courant.

4/ Ecrire une méthode

$$find(char * name)$$

retournant vraie ou faux si le répertoire de nom "*name*" se trouve dans le répertoire courant.

6/ Ajouter à la classe une méthode

$$Insertion(noeud * x, noeud * y)$$

permettant de créer le sous-répertoire x dans le répertoire d'adresse y. x étant supposé déjà créé.

7/ En déduire la méthode

$$mkdir(char * name)$$

permettant de créer, à partir de *current*, et **sans le changer**, un répertoire de nom "*name*", dans n'importe quel autre répertoire. La chaîne "*name*" doit pour cela être analysée de la manière suivante.

Un caractère spécial "/" permet d'indiquer un changement de répertoire. Il ne doit pas y avoir consécutivement plus de deux "/". Il faudra donc reformater, si nécessaire, la chaîne pour vérifier que cette régle est respectée.

Si le premier caractère lû est "/" cela veut dire qu'il faut se rendre à la racine. On enlevera ensuite ce premier caractère puis on lira la suite des mots séparés par "/". Si la chaîne ne commence pas par "/" alors on lit la suite des mots séparés par "/" sans se rendre au préalable à la racine.

Pour chaque mot lû (sauf le dernier), on cherche dans le répertoire précédemment rencontré, le sous-répertoire correspondant au mot lû. S'il existe, on s'y rend. Puis on réitére avec le mot suivant,etc. Le dernier mot lû devra être le nom du répertoire **à créer** dans **le répertoire précédemment rencontré**. On ne le créera cependant que s'il n'existe pas déjà.

8/ Ecrire une méthode

$$cd(char * name)$$

permettant de se rendre (si possible) à n'importe quel répertoire de nom "*name*" à partir du répertoire courant. C'est à dire d'affecter à "*current*" l'adresse du noeud correspondant. La chaîne "*name*" doit pour cela être analysée comme précédemment.

9/ Ecrire une méthode

$$Deplacer(noeud * r, noeud * y)$$

permettant de déplacer **tout** le répertoire de racine r vers le répertoire y.

10/ En déduire la méthode

$$mv(char * name, char * destination)$$

permettant de déplacer, à partir de *current*, et **sans le changer**, **tout** le répertoire "*name*" (s'il existe) vers le répertoire "*destination*" (s'il existe).

"*name*" et "*destination*" doivent aussi être analysés comme précédemment.

11/ Ecrire une méthode

$$new - mv(char * name, char * destination)$$

permettant de déplacer, à partir de "*current*", et **sans le changer**, **tous** les **sous-répertoires** de "*name*" vers "*destination*" puis de supprimer le noeud correspondant à "*name*".

12/ Ecrire, par le biais d'une méthode récursive de parcours en profondeur, une méthode

$$print(char * name)$$

affichant les chemins complets de tous les répertoires et sous-répertoires de "*name*".