Ce TP va vous permettre de manipuler des arbres n-aires en utilisant la notion d'arbre généalogique.

Nous ne travaillerons que sur des arbres généalogiques qui ont un ancêtre commun (la racine de l'arbre), qui aura un certain nombre d'enfants, et chaque enfant aura un certain nombres d'enfants ... On ne pourra donc remonter pour un individu que sur une branche de sa famille (père ou mère).

1 Construction d'un arbre

Vous allez écrire :

- une structure **noeud** qui aura 3 champs : une chaîne de caractère (le nom de la personne), un entier (le nombre d'enfants) et 1 pointeur sur un tableau de pointeurs de type "noeud" (pour accéder à chaque enfant).
- une fonction **createArbre :** qui prend en paramètre la Racine de l'arbre à créer **Racine**. Cette fonction :
  - construira un noeud qu'elle initialisera avec :
    - le nom de la personne saisi par l'utilisateur;
    - le nombre d'enfants saisi par l'utilisateur;
    - puis en fonction du nombre d'enfants, créera un tableau ayant autant de pointeurs que d'enfants;
  - puis construira les noeuds pour chaque enfant en rappelant cette fonction en passant chaque enfant comme la racine d'un nouvel arbre (sur le même principe que la fonction créant des arbres n-aires, sauf qu'il sera inutile de passer une profondeursur l'exemple du TP2, on passera une profondeur à l'arbre, ou lorsqu'on demande le nombre d'enfants qu'il est de 0 du coup on va remplir le noeud suivant).
  - La fonction s'arrêtera lorsque tous les enfants de la racine et leurs descendants propres seront saisis.

2 Affichage des arbres

Vous allez écrire une fonction qui permet d'afficher les noeuds d'un arbre à partir de la racine, en affichant le nom contenu dans le noeud et le nombre d'enfants. Puis vous relancerez l'affichage des sous-arbres ayant comme racine les enfants du noeud. L'affichage des noeuds se fera donc dans un ordre type préfixé où on affiche d'abord la racine avant tous les enfants et ceci récursivement.

Pour mieux voir les descendants de chaque noeud, vous rajouterez une tabulation (par rapport à l'affichage du noeud) pour l'affichage des enfants, et ceci récursivement. Ca revient à avoir devant l'affichage des caractéristiques du noeud un nombre de tabulations égal à la profondeur du noeud.

Exemple :

```
fred a 3 enfants
        yannis a 2 enfants
                toto a 1 enfants
                        titi a 1 enfants
                                tutu a 0 enfants
                tata a 0 enfants
        sami a 0 enfants
        noe a 1 enfants
                jaber a 0 enfants
```

3 Recherche dans un arbre

3.1 Recherche d'un ancêtre

Vous écrirez les fonctions suivantes :

- **rechercherParent** qui prend en paramètres la racine de l'arbre et le nom de la personne dont on veut rechercher le père (ou mère). La fonction renverra l'adresse du noeud. Le noeud sera NULL si la recherche n'a pas abouti.
- **rechercherGrandParent** qui prend en paramètres la racine de l'arbre et le nom de la personne dont on veut rechercher le grand-père (ou grand-mère). La fonction renverra l'adresse du noeud. Le noeud sera NULL si la recherche n'a pas abouti.

3.2 Recherche des caractéristiques

Ecrire les fonctions suivantes :

- **afficherCousins :** qui prend en paramètres la racine d'un arbre, et le nom de la personne dont on veut trouver les cousins. Pour celà vous utiliserez la fonction décrite précédemment "rechercherGrandParent" puisqu'un cousin est une personne qui a le même grand-parent que soi. Attention à ne pas écrire les frères et soeurs de la personne (qui ont aussi le même grand-parent).
- **afficherFreres :** qui prend en paramètres la racine d'un arbre, et le nom de la personne dont on veut trouver les frères et soeurs. Pour celà vous utiliserez la fonction décrite précédemment "rechercherParent".
- **rajouterEnfant :** qui prend en paramètres la racine d'un arbre, et le nom de la personne à qui on veut rajouter un enfant, avec des descendants éventuels.
  - Si vous êtes sur le bon noeud (dont le nom est celui passé en paramètre):
    - vous créerez un enfant (et ses descendants) en appelant la fonction "createArbre" qui permet de créer un sous-arbre.
    - vous rajouterez un enfant au bon noeud ("ptr"). Pour celà il faudra rajouter un noeud dans le tableau de pointeurs de ce noeud "ptr", il faudra donc recréer un nouveau tableau avec un noeud supplémentaire.
    - vous rajoutez un enfant donc n'oubliez pas d'incrémenter le nombre d'enfants du noeud "ptr".
  - Si vous n'avez pas atteint le bon noeud, alors rappelez la fonction sur ses descendants.

4 Programme principal

Vous écrirez un programme qui permet de tester les fonctions écrites précédemment:

- vous créerez un arbre ayant au moins une profondeur de 3, avec des feuilles de hauteur 1, 2 et 3.
- vous afficherez l'arbre
- vous rechercherez des cousins de noeuds ayant respectivement aucun cousin, ayant un ou plusieurs cousins, n'existant pas.
- vous rechercherez de la même façon des frères de noeuds ayant respectivement aucun frère, ayant un ou plusieurs frères, n'existant pas

5 Bonus

Vous écrirez une fonction **createArbreRoi** qui prend en paramètres un tableau de chaînes de caractères alternant des noms et le nombre d'enfants respectifs sous la forme :

```
grandpere 3
    enfant1 2
        petitenfant11 0
        petitenfant12 2
            arrierepetitenfant121 0
            arrierepetitenfant122 0
    enfant2 0
    enfant3 4
        petitenfant31 0
        petitenfant32 2
            arrierepetitenfant321 0
            arrierepetitenfant322 0
        petitenfant33 0
        petitenfant34 1
            arrierepetitenfant341 1
                  arrierearrierepetitenfant3411 0
```

Vous partirez de ce tableau représentant la descendance de Philippe IV Le Bel jusqu'à une profondeur de 4 :

string L[] = {"PhilippeIV", "7",
"LouisX", "2", "JeanneII", "8", "Jeanne", "0", "Marie", "4", "ConstanceDAragon", "0", "JeanneDAragon", "0", "MarieDAragon", "0", "PierreDAragon", "0", "BlancheDeNavarre", "0", "CharlesII", "8", "CharlesIII", "0", "Philippe", "0", "Marie", "0", "Pierre", "0", "Isabelle", "0", "Blanche", "0", "Jeanne", "0", "Bonne", "0", "AgnesDeNavarre", "1", "GastonDeFoix", "0", "PhilippeDeNavarre", "2", "LancelotBatard", "0", "RobineBatarde", "0", "Jeanne", "1", "CharlesDeRohan", "0", "LouisDeNavarre", "3", "CharlesDeBeaumont", "0", "TristanDeBeaumont", "0", "JeanneDeBeaumont", "0", "JeanIer", "0",
"MargueriteDeFrance", "0",
"BlancheDeFrance", "0",
"Isabelle", "4", "EdouardIII", "12", "EdouardDeWoodstock", "0", "Isabelle", "0", "Jeanne", "0", "William", "0", "LionelDAnvers", "0", "JeanDeGand", "0", "EdmondDeLanley", "0", "BlancheDeLaTourDeLondres", "0", "MarieDAngleterre", "0", "Marguerite", "0", "ThomasDeWindsor", "0", "GuillaumeDeWindsor", "0", "ThomasDeWoodstock", "1", "JeanDEltham", "0", "AlienorDeWoodstock", "2", "RenaudIII", "0", "Edouard", "0", "JeanneDeLaTour", "0",
"PhilippeV", "5", "JeanneIII", "1", "PhilippeMonsieur", "0", "MargueriteI", "1", "LouisIIDeFlandres", "0", "Isabelle", "0", "Blanche", "0", "Philippe", "0",
"CharlesIV", "3", "Jeanne", "0", "Marie", "0", "Blanche", "0",
"RobertDeFrance", "0" }

Pour convertir une chaîne de caractères en entier, il suffit d'appeler la fonction **stoi(string)** de la bibliothèque "<stream>" qui renvoie l'entier 4 si la chaîne est "4".