Corrigé du TD 7

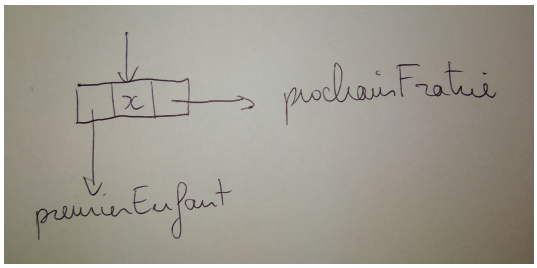
Exercice 1.

Question 1. *Rappelez comment représenter un arbre général ou une forêt sous la forme enfant-fratrie.*

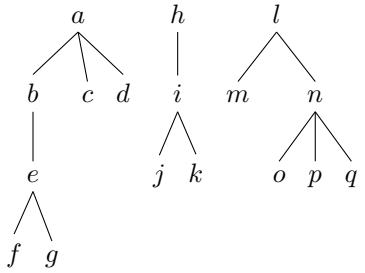
On utilise la structure et les types suivants :

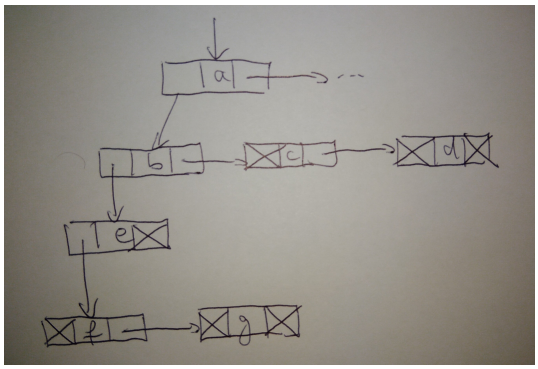
```
structure noeudF
    premierEnfant : pointeur sur noeudF
    valeur : entier
    prochainFratrie : pointeur sur noeudF

type forêt = pointeur sur noeudF
type arbreGénéral = pointeur sur noeudF
```

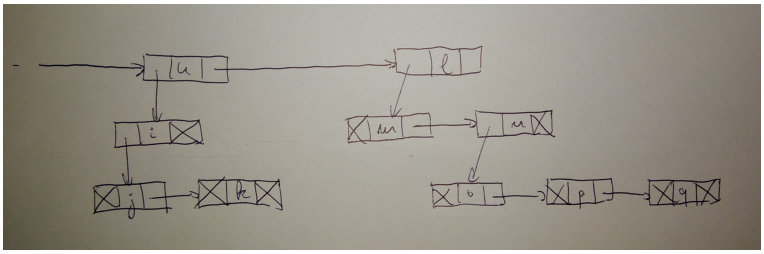
On peut représenter la structure `noeudF` de la manière suivante :



On peut alors représenter la forêt en entier comme suit :







Question 2. *Écrire une procédure qui renvoie le nombre de nœuds internes d'une forêt.*

Rappel de la procédure analogue pour les arbres binaires :

```
nombreInternes(A : arbreBinaire)
    # cas de l'arbre vide :
    si A == None alors retourner 0
    # cas d'une feuille :
    si A->gauche == None et A->droit == None alors retourner 0
    # cas d'un noeud interne
    retourner 1 + nombreInternes(A->gauche) + nombreInternes(A->droit)
```

Cas d'une forêt :

```
nombreNoeudsInternes(F : forêt) : entier
    # cas de l'arbre vide
    si F == None alors retourner 0
    # cas d'une feuille
    si F->premierEnfant == None alors retourner nombreNoeudsInternes(F->prochainFratrie)
    # cas d'un noeud interne
    retourner 1 + nombreNoeudsInternes(F->premierEnfant)
                + nombreNoeudsInternes(F->prochainFratrie)
```

Question 3. *Soit* `N` *un nœud de profondeur* `k`*. Quelle est la profondeur de* `N->premierEnfant` *et* `N->prochainFratrie` *? Écrire une procédure qui renvoie le nombre de nœuds de niveau* `k` *dans une forêt* `F`*.*

Si la profondeur de `N` est `k`, celle de `N->premierEnfant` est `k+1` et celle de `N->prochainFratrie`, `k`.
Si on cherche les nœuds de profondeur `k` à partir de `N`, en passant à `N->prochainFratrie` on cherche toujours les nœuds de profondeur `k`, par contre en passant à `N->premierEnfant` on cherche les nœuds de profondeur $k-1$.

```
nombreNoeudsNiveau(F : forêt, k : entier) : entier
    # cas de la forêt vide
    si F == None alors retourner 0
    # cas où on a atteint le niveau recherché
    # toute la fratrie sera également du bon niveau
    si k == 0 alors
        retourner 1 + nombreNoeudsNiveau(F->prochainFratrie,0)
    # sinon on continue à parcourir la forêt
    retourner nombreNoeudsNiveau(F->prochainFratrie,k) +
              nombreNoeudsNiveau(F->premierEnfant,k-1)
```

Question 4. *Écrire une procédure qui teste si deux forêts sont égales (mêmes nœuds et mêmes valeurs). Déterminer le coût de la procédure, défini comme étant le nombre de comparaisons de la valeur d'un nœud de la première forêt avec celle d'un nœud de la deuxième.*

```
égalesForêts(F, G : forêt) : booléen
    # cas de deux forêts vides
    si F == None et G == None alors retourner Vrai
    # cas où une forêt est vide et pas l'autre
    # (on a déjà exclu le cas où les deux forêts sont vides...)
    si F == None ou G == None alors retourner Faux
    # cas où les racines ont des valeurs différentes
    si F->valeur != G->valeur alors retourner Faux
    # on compare la suite de l'arbre récursivement
    retourner égalesForêts(F->premierEnfant, G->premierEnfant) et
              égalesForêts(F->prochainFratrie, G->prochainFratrie)
```

Voici une version concise avec les opérateurs booléens `et` et `ou` :

```
égalesForêts(F, G : forêt) : booléen
    retourner ( F == None et G == None ) ou
              ( F <> None et G <> None et F->valeur == G->valeur et
                égalesForêts(F->premierEnfant, G->premierEnfant) et
                égalesForêts(F->prochainFratrie, G->prochainFratrie) )
```

Le coût dans le pire des cas est égal au nombre de nœuds de la plus petite forêt (cas où les deux arbres sont égaux, ou différents uniquement au dernier nœud visité). Le coût dans le meilleur des cas vaut 1 (valeurs différentes pour la première racine).

Question 5. *Expliquer comment parcourir la branche la plus à gauche d'une forêt. Quelle est la différence avec le cas des arbres binaires ? Écrire une procédure qui affiche cette branche. Procéder de même avec la branche la plus à droite.*

Il suffit de suivre les pointeurs `F->premierEnfant`. C'est plus simple que dans le cas des arbres binaires où il faut suivre `A->gauche`, mais également `A->droit` si `A->gauche == None`.

```
brancheGauche(F : forêt)
    si F == None alors retourner
    afficher F->valeur
    brancheGauche(F->premierEnfant)
```

Pour la branche droite en revanche c'est plus compliqué car on doit suivre `F->premierEnfant` puis parcourir la fratrie jusqu'au dernier élément.

```
brancheDroite(F : forêt)
    # cas de la forêt vide
    si F == None alors retourner
    # on parcourt la fratrie jusqu'au bout
    tant que F->prochainFratrie <> None:
        F = F->prochainFratrie
    # on affiche la valeur
    afficher F->valeur
    # on descend dans l'arbre
    brancheDroite(F->premierEnfant)
```

Question 6. *On veut calculer le degré d'un arbre général, c'est-à-dire le degré maximal de ses nœuds. Étant donné un nœud* `N`*, on appelle* `d` *le nombre de membres de la fratrie de* `N` *déjà visités (y compris* `N`*). Comment évolue* `d` *quand on suit* `N->premierEnfant` *et* `N->prochainFratrie` *? En déduire une procédure récursive qui renvoie le degré d'un arbre général.*

Quand on suit `N->premierEnfant` on a `d = 1`, et quand on suit `N->prochainFratrie`, `d=d+1`.

```
degréArbre(A : arbreGeneral, d : entier) : entier
    si A == None alors retourner d-1
    retourner max(degréArbre(A->prochainFratrie, d+1),
                  degréArbre(A->premierEnfant,1))
```

On appelle cette procédure par `degréArbre(A,1)`.
Voici une variante où on s'impose de ne pas faire d'appel récursif sur l'arbre vide :

```
degréArbre2(A : arbreGeneral, d : entier) : entier
    r : entier ; r = d
    si A->prochainFratrie <> None alors
        r = degréArbre2(A->prochainFratrie, d+1)
    si A->premierEnfant <> None alors
        r = max(r, degréArbre2(A->premierEnfant,1))
    retourner r
```

Avec cette nouvelle version, chaque nœud non vide est visité exactement une fois. Il y a autant d'appels de la fonction que de nœuds de l'arbre, indépendamment de la structure de celui-ci.

Question 7.

Voici une procédure d'affichage en largeur.
On crée un premier nœud « vide » `N` pour se ramener au cas d'un arbre général.

```
affichageLargeur(F : forêt)
    N : forêt ; N = Nouveau(noeudF) ; N->valeur = ''
    N->prochainFratrie = None ; N->premierEnfant = F
    f : file ; f = initFile() ; f = enfiler(f, N)
    tant que non fileVide(f) faire
```

```
        F = tete(f) ; f = defiler(f)
        afficher F->valeur
        F = F->premierEnfant
        tant que F <> None faire
            f = enfiler(f, F)
            F = F->prochainFratrie
```

La procédure suivante réalise l'affichage dans l'ordre préfixe en profondeur.

```
affichagePréfixe (F : forêt)
    si F == None alors retourner
    afficher F->valeur
    affichagePréfixe(F->premierEnfant)
    affichagePréfixe(F->prochainFratrie)
```

Pour l'affichage suffixe il suffit de déplacer l'instruction `afficher F->valeur` entre les deux appels récursifs.