Arbres binaires

Dans ce TD, qui durera plusieurs séances, on va construire pas à pas nos premières fonctions sur les arbres.

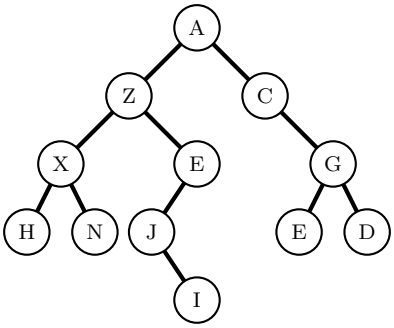

FIGURE 1 – Arbre binaire

1. **Vocabulaire** — Dans cet exercice, on s'intéressera exclusivement à l'exemple de l'arbre donné en Figure 1.
   a. Donner les étiquettes
      — de la racine ;
      — des feuilles ;
      — des nœuds de hauteur 2 ;
      — des nœuds de niveau 3 ;
      — des nœuds sur le chemin A → J.
   b. Donner le nombre de nœuds, le nombre de nœuds internes et le nombre de feuilles.
   c. Donner les étiquettes de nœuds visités lors d'un parcours en profondeur avec traitement :
      — préfixe (on affiche la racine, puis le sous-arbre gauche, puis le sous-arbre droit) ;
      — infixe (on affiche le sous-arbre gauche, puis la racine, puis le sous-arbre droit) ;
      — suffixe (on affiche le sous-arbre gauche, puis le sous-arbre droit, puis la racine).
   d. Donner les étiquettes des nœuds visités lors d'un parcours en largeur.
   e. On rappelle que la hauteur d'un arbre est le nombre maximum de liens entre la racine et une feuille. Donner la hauteur d'un arbre réduit à une seule feuille, puis d'un arbre vide, et enfin de l'arbre ci-dessus.
      Quelle est la hauteur du nœud d'étiquette Z ?

2. **Programmation** — Dans cet exercice et les exercices qui suivront, on utilisera les types C suivants :

```
typedef struct noeud{
   int valeur;        /* étiquette */
   struct noeud *fg;  /* fils gauche */
   struct noeud *fd;  /* fils droit */
} Noeud, *Arbre;
```

   a. Écrire une fonction renvoyant la hauteur d'un arbre.
   b. Écrire une fonction renvoyant le nombre de nœuds d'un arbre.
   c. Écrire une fonction renvoyant le nombre de nœuds internes d'un arbre.
   d. Écrire une fonction renvoyant le nombre de feuilles d'un arbre.
   e. Écrire une fonction renvoyant le nombre de nœuds de l'arbre qui possèdent exactement deux fils.

3. **Parcours** — Écrire une fonction affichant les étiquettes des nœuds d'un arbre lors d'un parcours en profondeur

   — dans l'ordre préfixe ;
   — dans l'ordre infixe ;
   — dans l'ordre suffixe.

   Rappeler la définition d'une file (ainsi que les opérations possibles) et donner l'algorithme permettant d'afficher un arbre binaire par un parcours en largeur.

4. **Description d'un arbre étiqueté par des entiers strictement positifs** — Dans cet exercice, on souhaite afficher les nœuds dans l'ordre d'un parcours en profondeur préfixe : on représente chaque nœud de l'arbre par son étiquette, et un fils vide est représenté par l'entier `0`.

   a. Construire les deux arbres binaires décrits par les deux séquences suivantes :
      — 8321005007009400600
      — 434520001300003500210060 0
   b. Écrire la fonction `void afficheArbPositif(Arbre a)` qui affiche la suite des entiers décrivant un arbre.
   c. Décrire la methode pour construire un arbre binaire à partir d'une suite d'entiers positifs (on pourra supposer que cette suite est valide, c'est-à-dire qu'elle représente bien un arbre binaire).
   d. Écrire la fonction `int construitArbPositif(Arbre *a)` qui construit un arbre binaire à partir d'une suite d'entiers valide (saisie au clavier). La fonction stockera cet arbre à l'adresse vers laquelle pointe le pointeur `a` ; elle renverra `0` en cas d'erreur, `1` en cas d'exécution correcte.

5. **Arbre héréditairement gauche** — Un arbre strictement binaire (c'est-à-dire un arbre ou chaque noeud a 0 ou 2 fils) est dit *héréditairement gauche* s'il est réduit à sa racine, ou bien s'il vérifie les trois conditions suivantes :

   — le sous-arbre gauche a un nombre de feuilles supérieur ou égal à celui de droite ;
   — le sous-arbre gauche est héréditairement gauche ;
   — le sous-arbre droit est héréditairement gauche.

   a. Tracer tous les sous-arbres héréditairement gauche à trois feuilles, puis à cinq feuilles.
   b. Écrire une fonction `int estHG(Arbre a)` qui renvoie `1` si l'arbre `a` (supposé strictement binaire) est héréditairement gauche, et qui renvoie `0` sinon.