Exercice 1 : L'arbre figure 1 représente une opération arithmétique.

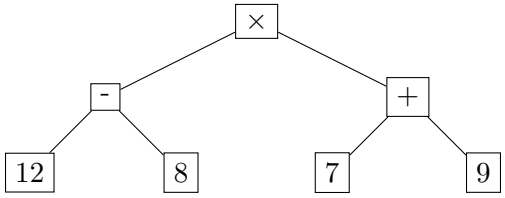
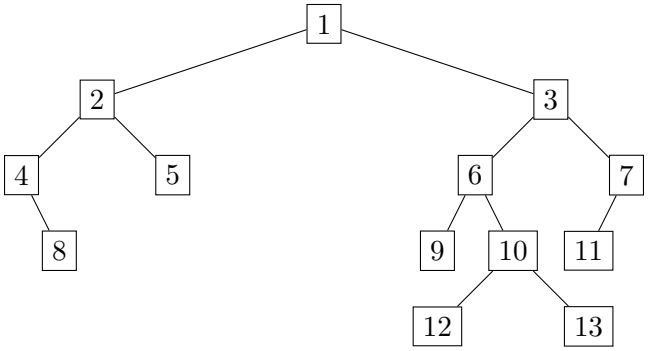


FIGURE 1 – Calcul arithmétique

1. Parcourir cet arbre en profondeur (préfixe, infixe et postfixe).
2. Donner le résultat du calcul.
3. Pour quel parcours est-il indispensable de rajouter des parenthèses ?

Exercice 2 :



1. Effectuer les parcours (largeur, profondeur) sur l'arbre binaire.
2. Quelle est la hauteur de cet arbre. On considère qu'un arbre vide à une hauteur de -1.
3. Cet arbre est-il équilibré ?
4. Cet arbre est-il complet ?

Exercice 3 :

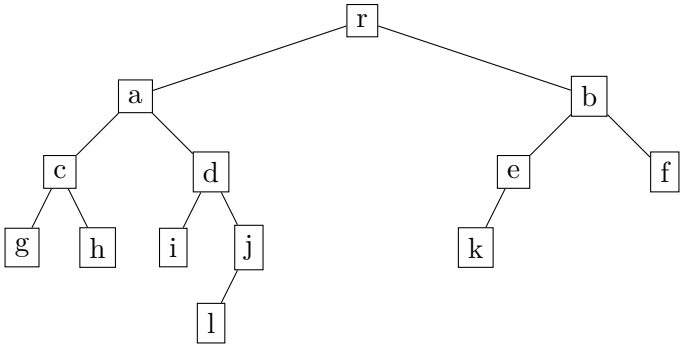


FIGURE 2 – Un arbre de chaîne de caractère

L'objectif est de créer une classe permettant de construire un arbre binaire de chaînes de caractères distinctes.

1. Créer la classe **`Arbre_binaire`** et son constructeur. On passera un paramètre **`h`** qui initialisera l'attribut **`hauteur`** de l'arbre. Le constructeur initialisera :
   — un tableau **`arbre`** rempli d'objet **`None`** de la taille de l'arbre binaire parfait correspondant à **`h`**.

Christophe Viroulaud

— la racine de l'arbre avec la chaîne de caractère `"r"`.

2. Écrire la méthode `get_taille(self)` → `int` qui renvoie le nombre de nœuds de l'arbre.
3. Écrire la méthode `get_indice(self, chaine: str)` → `int` qui renvoie la position de la chaîne dans le tableau.
4. Écrire la méthode `inserer(self, pere: str, gauche: str, droit: str)` → `None` qui ajoute les fils `gauche` et `droit` au nœud `pere`. La méthode lèvera une erreur d'assertion si le nœud `pere` ne peut pas avoir de fils (sort du tableau).
5. Écrire la méthode récursive `prefixe(self, position: int, parcours: list)` → `None` qui effectue un parcours préfixe et complète le tableau `parcours` au fur et à mesure.
6. Écrire sur le même modèle les méthodes `infixe` et `postfixe`
7. **Pour les plus avancés :** Écrire la méthode récursive `prefixe_2(self, position: int)` → `list` qui construit (par concaténation) le tableau de parcours.

Christophe Viroulaud