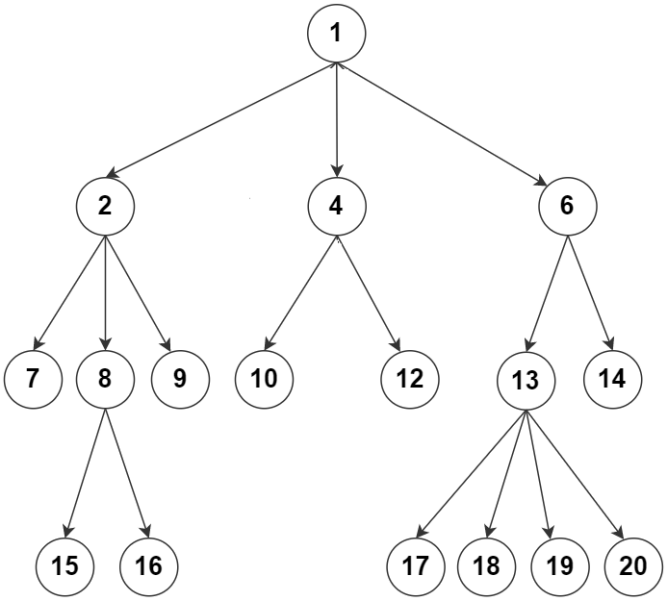
TD 04 : Arbres

Consignes générales : N'oubliez pas pour ce TD comme pour les suivants de vous créer un répertoire consacré au TD et d'enregistrer vos codes dedans.

On rappelle que les commandes à taper dans le terminal pour compiler puis éxécuter votre programme C :
— Pour compiler : *gcc -o nom_executable nom_programme.c*
— Pour exécuter : *./nom_executable*

Exercice 1 (*Question de cours*)
Soit l'arbre suivant :

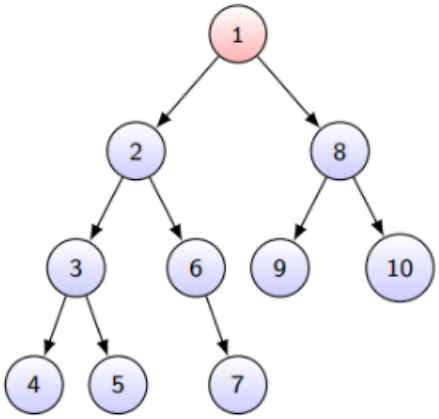


1. Donner :
   — l'ordre de l'arbre
   — le degrés du nœud 4 et du nœud 2
   — le nombre de feuilles
   — la hauteur de l'arbre
2. Quel est le parcours en longueur préfixe de cet arbre ?
3. Quel est le parcours en largeur de cet arbre ?

Exercice 2 (*Arbre binaire*) Remarque : la plupart des fonctions démandées dans cet exercice sont disponibles en pseudo-code dans le cours.

1. Construction de l'arbre :
   (a) Déclarer une structure `Arbre` permettant de gérer un arbre binaire contenant des entiers.
   (b) Redéfinir le type pointeur sur arbre en `pArbre`.
   (c) Créer la fonction `creerArbre` permettant de creer un un arbre. Cette fonction prend en paramètre l'élément à insérer dans le nœud de l'arbre, initialise ses fils et retoune son adresse.
   (d) Écrire la fonction `int estVide(pArbre a)` qui retourne 1 si a est un arbre vide, 0 sinon.
   (e) Écrire la fonction `int estFeuille(pArbre a)` qui retourne 1 si l'arbre est une feuille, 0 sinon.
   (f) Écrire la fonction `int element(pArbre a)` qui retourne l'élément stocké dans le nœud de a.
   (g) Écrire une fonction `exitseFilsGauche(pArbre a)` qui retourne 1 si l'arbre a un fils gauche, 0 sinon. Faire une fonction similaire `exitseFilsDroit(pArbre a)`.
   (h) Écrire une fonction `ajouterFilsGauche(pArbre a, int e)` qui créé à l'arbre un fils gauche qui va contenir `e`. Faire de même avec `ajouterFilsDroit(pArbre a, int e)`.

(i) A l'aide des fonctions réalisées précédement, construire l'arbre suivant :



2. Parcours de l'arbre :
   (a) Écrire une fonction `traiter` qui affichera le contenu du nœud de l'arbre passé en paramètre.
   (b) Écrire une procédure `parcoursPrefixe` permettant d'afficher tous les éléments d'un arbre par un parcours en profondeur préfixe.
   (c) Vérifier que l'arbre que vous avez construit est correct en affichant son parcours préfixe.
   (d) Écrire une procédure `parcoursPostfixe` permettant d'afficher tous les éléments d'un arbre par un parcours en profondeur postfixe. Tester la fonction sur l'arbre.
   (e) Déclarer une structure permettant de gérér une **FILE** contenant des `pArbre`.
   (f) Écrire une procédure `parcoursLargeur` permettant d'afficher tous les éléments d'un arbre par un parcours en largeur.
3. Modification de l'arbre
   (a) Écrire une fonction `parbre modifierRacine(pArbre a, int e)` qui modifie l'élément stocké dans `a` par l'élément `e` et retourne `a`.
   (b) Écrire une fonction `pArbre supprimerFilsGauche(pArbre)` qui supprime le fils gauche d'un arbre. Idem avec `supprimerFilsDroit(pArbre)`. **Attention aux fuites mémoire !**(voir cours).
   (c) Supprimer les nœud 9, 15 et 3 de l'arbre. Quel devrait être son parcours en largeur ? Vérifier.
4. Annalyse de l'arbre
5. Écrire une fonction `nmbFeuille` qui retourne le nombre de feuilles d'un arbre.
6. Écrire une fonction `tailleArbre` qui retourne la taille (nombre de nœuds internes) de l'arbre.
7. *pas facile !* Écrire une fonction `hauteur` qui retourne la hauteur d'un arbre. Elle renvera -1 si l'arbre est vide.

Exercice 3 (*Arbre binaire filiforme*)

Un arbre binaire est dit *filiforme* si chaque nœud a au plus un seul fils (qu'il soit gauche ou droit).

— A quoi va ressembler un tel arbre ?
— Écrire une fonction permettant de determiner si un arbre est filiforme (plusieurs méthodes sont possibles !).
— Un arbre est dit *peigne gauche* si les nœuds n'ont qu'un fils gauche et pas de fils droit.
  — Écrire une fonction permettant de determiner si un arbre est peigne gauche.
  — Écrire une fonction `parbre constrPeigneGauche(int h)` qui va créer un arbre peigne gauche de hauteur h en le remplissant avec des valeurs aléatoires entre 0 et 10. L'afficher.

Exercice 4 (*Notation polonaise inversée (devoir de 2021-2022)*)

La *notation polonaise inversée* (NPI) (utilisée par les calculatrices Texas Instrument entre autres) permet d'écrire de façon non ambiguë les formules arithmétiques sans utiliser de parenthèses. Le principe est que les opérandes précèdent toujours leurs opérateurs et peuvent être eux-mêmes des nombres ou des expressions non triviales.

Exemples $1 + 2$ s'écrit en notation polonaise $12+$
$((1 + 2) * 4) + 3$ s'écrit $12 + 4 * 3+$
$\frac{a}{a+b}$ s'écrit $aab+/$

Dans cet exercice, nous abordons comment construire une notation polonaise à l'aide d'arbre binaire. Chaque nœud de cet arbre contiendra une opérande ou un nombre.

On considère les structures suivantes :

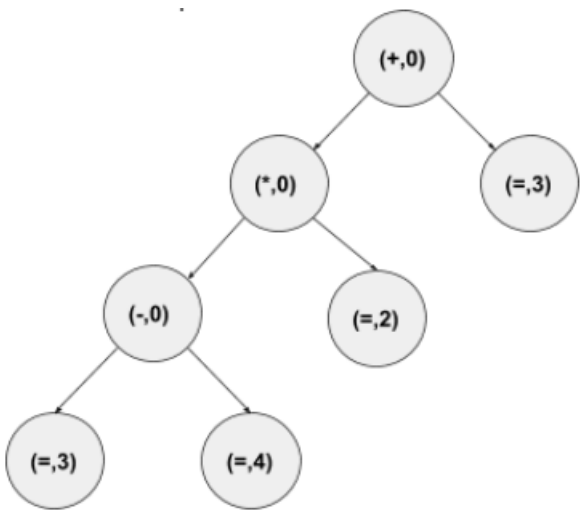
```
Struture Terme

    typeTerme : Caractere
    valeur : Reel

Structure Arbre

    terme : Structure Terme
    fg, fd : Pointeur sur Structure Arbre
```

Le champs `typeTerme` survira à définir les opérandes ; il ne pourra prendre que les valeurs suivantes : '+', '-', 'x' , ' /' et '='.

Si `typeTerme` vaut '=' c'est que le nœud permet de stocker un nombre. Si ce n'est pas le cas, le champs `valeur` vaudra 0.

Chaque nœud interne contient un opérateur binaire et son fils à gauche et son fils à droite sont respectivement l'expression à gauche de l'opérande et l'expression à droite de l'expression. Les feuilles sont donc forcément des constantes.

1. Quelle est l'écriture en NPI du calcul suivant ?
   $5 * \dfrac{(a+1)*(b+1)}{a+b}$
2. Quelle est la valeur de l'expression polonaise inversée suivante : 1 8 2 – 7 4 – * 3 6 + / / ?
3. L'expression $((3-4)*2+3)$ s'écrit sous forme d'arbre comme suit :



La notation polonaise inverse de ce calcul est $34-2*3+$

À quel type de parcours d'arbre correspond la notation polonaise inversée ?

4. A partir des structures définies ci-dessus et des fonctions écrites lors des exercices précédents, construire l'arbre représenté ci-dessus.
5. Écrire la procédure `void afficherNotationPolonaiseInversee (pArbre a)` qui permet d'afficher en notation polonaise inversée l'expression mathématique définie dans l'arbre binaire.
6. Écrire la fonction `float eval (pArbre a)` qui permet donner le résultat de l'expression mathématique définie dans l'arbre binaire. La tester avec l'arbre puis pour vérifier votre réponse à la question 2.