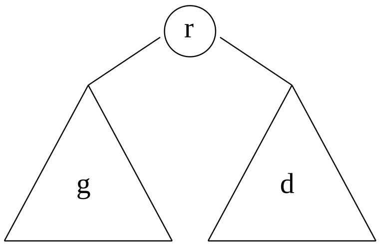
Parcours d'un arbre binaire

Rappel : un arbre binaire T est défini récursivement dans la façon suivante :
- il existe un arbre vide (c'est celui qui n'a pas de noeuds). T peut être celui ci.
- ou T est défini par : un noeud r appelé racine, constitué :
  - d'un label ou étiquette, (ici on prendra des caractères)
  - d'un arbre binaire g appelé sous-arbre gauche,
  - et d'un arbre binaire d appelé sous-arbre droit.

On peut représenter graphiquement T comme dans la figure suivante :



Un arbre binaire peut être implementé avec les classes suivantes :

```
class Noeud{
  char etiquette;
  Arbre gauche;
  Arbre droite;
  Noeud(char c, Arbre x, Arbre y){
    etiquette=c;
    gauche=x;
    droite=y;
  }
}

class Arbre{
  Noeud n;
  Arbre (Noeud x){
    n=x;
  }
}
```

Dans cette présentation on a choisi de raffiner ce qu'on faisait sur les listes. Vous vous rappellez que la liste vide était celle qui valait `null`. A présent on distingue les arbres non déclarés, de ceux qui n'ont pas de noeuds. Ainsi

```
Arbre a=null;
Arbre b=new Arbre(null);
Arbre c=new Arbre(null);
Arbre d=new Arbre('x',b,c);
```

sont respectivement un arbre non déclaré, un arbre vide, encore un arbre vide, et un arbre à un seul noeud.

Exercice 1 (TP-TD). Parmis les constructions suivantes, préciser lesquelles ne sont pas définies, lesquelles désignent des arbres vides, des arbres bien formés, ou des constructions valides pour le compilateur mais incorrectes pour les spécifications. Puis modifier (ou ajouter) les constructeurs pour faire en sorte de compléter naturellement les constructions incomplètes.

```
Arbre a=new Arbre();
Arbre b=new Arbre(null);
Arbre c=b;
Arbre d;
Arbre e=new Arbre(new Noeud('x',b,c));
Arbre f=new Arbre(new Noeud('y',e,d));
Arbre g=new Arbre ('z');
```

Exercice 2 (TP). Ecrivez un programme court avec dans le main quelques déclarations d'arbres.

Parcours d'un arbre : algorithmes de base

motivation Lorsque nous nous interessions aux listes, pour les afficher par exemple, il nous suffisait de suivre le fil défini de proche en proche par les successeurs. Le premier élément d'une liste était evidement la tête de liste, le second son suivant etc ...
A présent, pour les arbres, comment savoir lequel est le premier le second ou le troisieme ?

La levée de cette indécision conduit à plusieurs parcours d'arbres. On en présente ici quelques uns très classiques.

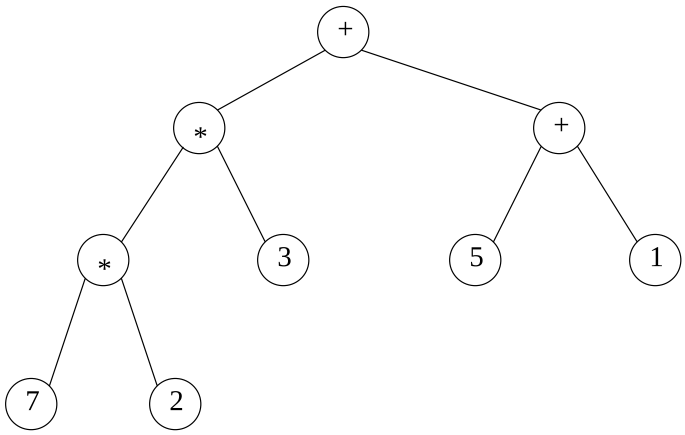
Le parcours d'un arbre en `ordre préfixe` est défini comme suit :
- rendre visite à la racine ;
- visiter le sous-arbre gauche en `ordre préfixe` ;
- visiter le sous-arbre droit en `ordre préfixe`.

Le parcours d'un arbre en `ordre suffixe` est défini :
- visiter le sous-arbre gauche en `ordre suffixe` ;
- visiter le sous-arbre droit en `ordre suffixe` ;
- rendre visite à la racine.

Le parcours d'un arbre en `ordre infixe` est défini :
- visiter le sous-arbre gauche en `ordre infixe` ;
- rendre visite à la racine ;
- visiter le sous-arbre droit en `ordre infixe`.

Note : Remarquez la récursivité de ces définitions.

Exercice 3 (TD). Pour l'arbre de la figure suivante, donner la séquence des étiquettes quand l'arbre est visité, respectivement, en ordre préfixe, suffixe ou infixe.



Exercice 4 (TD). Ecrire une méthode `statique` qui prend en entrée un arbre et affiche en sortie la séquence des étiquettes quand les sommets sont visités en ordre préfixe.

Exercice 5 (TD). Expliquez comment la distinction qu'on a faite, entre arbre vide et arbre non défini, permet d'ecrire une méthode `dynamique` qui fasse la meme chose que la précédente, et qui soit valable aussi sur les arbre vide.

Exercice 6 (TP). Ecrire une fonction qui prend en entrée un arbre (ou, plus précisement sa racine) et affiche en sortie la séquence des étiquettes quand les sommets sont visités en ordre infixe.

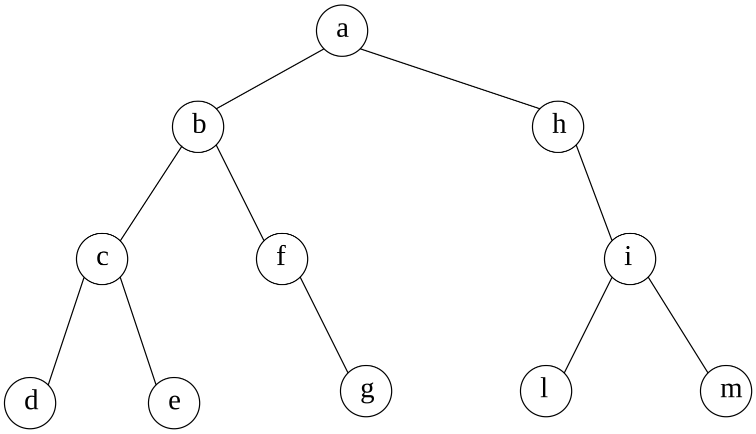
Autres algorithmes Les stratégies vues jusqu'ici sont, par leur nature, réalisées par des algorithmes récursifs. On veut maintenant formuler des algorithmes non récursifs pour decrire d'autres stratégies de parcours. Nous utiliserons ici des structures de données dejà vues pour stocker les sommets qui sont en attente d'être visités. On s'aperçoit qu'avec des structures de données différentes on obtient des stratégies differentes de parcours.

On considère l'algorithme suivant, où les sommets sont placés dans une liste de noeuds ; T représente l'arbre donné et r est sa racine :

Parcours $(T; r)$
- on initialise une liste L en n'y plaçant que la racine ;
- à chaque itération on "choisit" un noeud dans la liste L, et on le supprime.On l'affiche, puis ses deux fils sont "placés" dans la liste ;
- on continue jusqu'à ce que la liste soit vide.

Exercice 7 (TD). Que se passe-t-il si la liste L est gérée comme une *pile* ? Quel est l'ordre de visite des sommets sur l'exemple de la figure suivante ?

Exercice 8 (TD). Que se passe-t-il si la liste L est gérée comme une *file* (first-in-first-out) ? Quel est à présent l'ordre de visite des sommets ?



Exercice 9 (TP). Ecrire un programme qui prenne en entrée la racine d'un arbre et renvoie la liste des sommets dans l'ordre donné par l'algorithme **Parcours**(T, r) quand la liste est gérée comme une pile.

Exercice 10 (TP). Ecrire un programme qui prenne en entrée la racine d'un arbre et renvoie la liste des sommets dans l'ordre donné par l'algorithme **Parcours**(T, r) quand la liste est gérée comme une file.

Les Tas Les structures de données chainées sont souvent mise en opposition avec les tableaux. C'etait clair pour les listes, en voici une illustration pour les arbres, ou plutot pour certains types d'arbres.

Exercice 11 (TP-TD). Soit A un tableau de taille n. A partir de A, construire un arbre en suivant les regles récursives :
- la racine est étiquetée par $A[0]$;
- si un noeud v est étiqueté par $A[i]$ et $2i + 1 \leq n$, alors v a un fils gauche étiqueté $A[2i + 1]$;
- si un noeud v est étiqueté par $A[i]$ et $2i + 2 \leq n$, alors v a un fils droit étiqueté $A[2i + 2]$.

Un arbre de ce type s'appelle *tas*. Quels peuvent être les avantages de l'utilisation d'un tas trié dans un algorithme de tri ? Ecrire un programme qui prend en entrée un tableau et construit l'arbre correspondant.