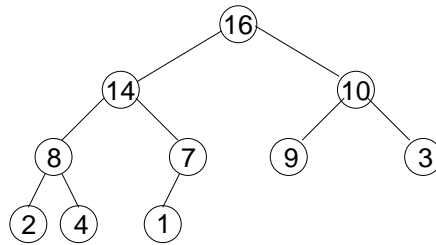


## Les arbres et la structure de *tas*

Dans ce TP, nous nous intéressons à des arbres binaires particuliers, où toutes les feuilles sont sur au plus deux niveaux différents, et où les feuilles les plus profondes sont à gauche. Par exemple,



De plus, nous supposons que l'arbre satisfait la *propriété de décroissance* : chaque nœud a une valeur supérieure à celle de son fils droit et à celle de son fils gauche (comme sur l'exemple).

### 1ère partie : la représentation par tas

On peut représenter un tel arbre par un tableau, appelé un *tas* : les nœuds de l'arbre sont énumérés niveau par niveau, la racine en premier, puis ses deux fils, etc. L'arbre de l'exemple est ainsi représenté par le tableau

16	14	10	8	7	9	3	2	4	1
----	----	----	---	---	---	---	---	---	---

Un des intérêts de la structure de tas est que l'on peut accéder aux fils ou au père d'un nœud facilement. En effet, si le tableau est indexé à partir de 1, le nœud d'indice  $i$  a pour fils gauche le nœud d'indice  $2 * i$ , pour fils droit celui d'indice  $2 * i + 1$ , et pour père celui d'indice  $i/2$ . Vérifiez sur l'exemple précédent.

**Question 1.** Écrire la spécification d'un paquetage `paq_tas` pour la manipulation de tas dont les éléments sont de type *générique discret* `Element`. Pour la représentation d'un tas, on utilise un enregistrement à discriminant, plutôt qu'un simple tableau.

```

type Tas(Taille:Natural:=0) is record
  Nbre_element : Natural:=0;
  Valeurs      : ... ;
end record;
  
```

Qu'est-ce qui justifie ce choix ?

**Question 2.** Ecrire des procédures d'entrées-sorties pour le type `Tas`.

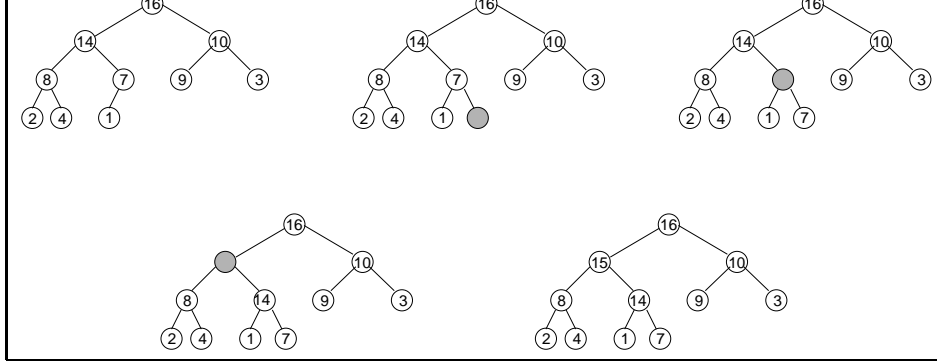
**Question 3.** Ajouter au paquetage les fonctions `Pere`, `Fils_droit` et `Fils_gauche`, qui déterminent respectivement l'indice du père, du fils droit et du fils gauche de l'élément dont l'indice est passé en argument. Pensez à gérer proprement les situations pathologiques.

**Question 4.** Ecrire une fonction `Est_feuille` qui teste si un indice du tas correspond à une feuille dans l'arbre sous-jacent. Remarquer qu'une feuille est un élément qui n'a pas de fils gauche.

**Question 5.** Ecrire une procédure `Inserer`, qui insère un nouvel élément dans un tas, en préservant bien sûr la propriété de décroissance. Pour cela, il faut

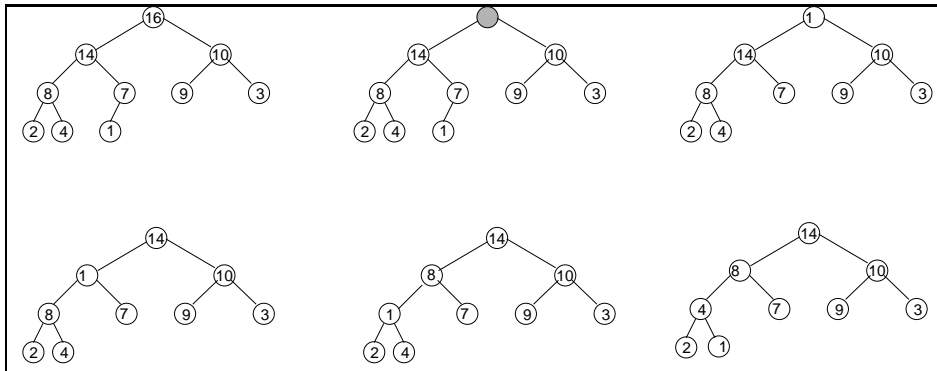
1. ajouter un nouvel élément au tas, en fin,
2. faire remonter le nouvel élément jusqu'à sa place définitive, en utilisant la procédure `Pere`.

La figure au dos détaille ces étapes, pour l'insertion de la valeur 15.



**Question 6.** Ecrire une procédure `Extrait_max(T: in out Tas; E: out Element)`, qui supprime du tas T l'élément maximal, le stocke dans E, et réorganise le tas. Les étapes à suivre sont :

1. lire la valeur maximale et la stocker dans E,
2. mettre la dernière valeur en tête,
3. supprimer la dernière feuille,
4. faire descendre la première valeur, à droite ou à gauche, jusqu'à ce que ses deux fils soient plus petits, ou que l'on ait atteint l'extrémité d'une branche.



## 2ème partie : un nouvel algorithme de tri

Une des applications privilégiées de la structure de tas est la définition d'un algorithme de tri, procédant en deux étapes :

- à partir de l'ensemble à trier, construire un tas, avec la procédure `Inserer`,
- à partir du tas contenant tous les éléments, construire la liste triée par applications successives de la procédure `Extrait_max`

L'avantage de cet algorithme est double. Premièrement, la complexité obtenue est tout à fait satisfaisante (en  $n \log(n)$ ). D'autre part, l'algorithme ne nécessite qu'un parcours séquentiel de l'ensemble à trier, ce qui fait qu'il se prête bien au tri de listes chaînées.

**Question 7.** Ecrire un programme de tri par ordre croissant pour les listes d'entiers, basé sur l'algorithme ci-dessus. Pour cela, posez-vous les questions : comment vais-je décrire la liste initiale et construire le tas ? Une fois le tas construit, comment vais-je construire la liste triée ?