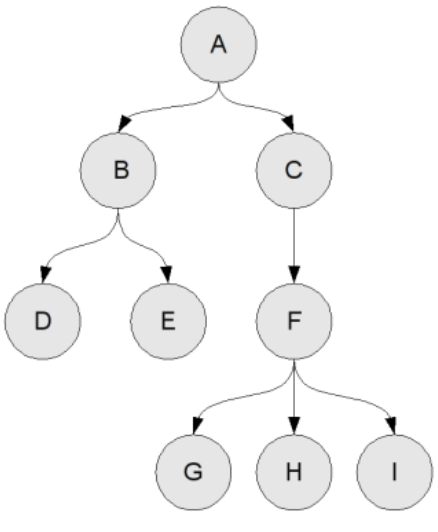
# ARBRE BINAIRE

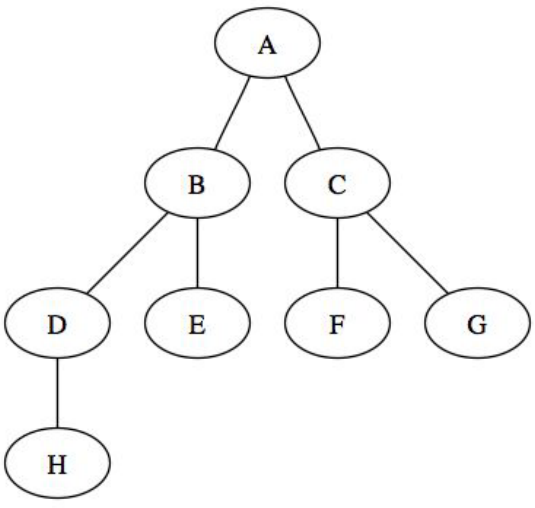
## DEFINITION ARBRE

C'est une structure très utilisée en informatique.
On peut considérer un arbre comme une généralisation d'une liste.
Cette structure contient des noeuds pouvant avoir des enfants (noeuds).



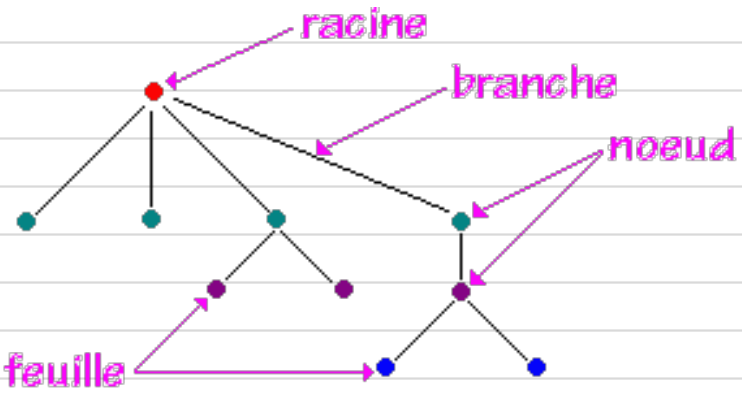

## DEFINITION ARBRE BINAIRE

C'est un arbre dont chaque noeud peut comporter au plus deux fils.
On connaît donc son degré maximum qui est de deux.




## Deux types de noeuds particuliers :

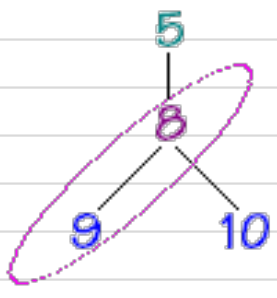
- Feuilles ➡️ ce sont des noeuds n'ayant pas d'enfant.
  On peut dire que ce sont les derniers noeuds de l'arbre.
- Racine ➡️ c'est un noeud n'ayant pas de parent.
  On peut dire que c'est le point de départ de l'arbre donc la racine.

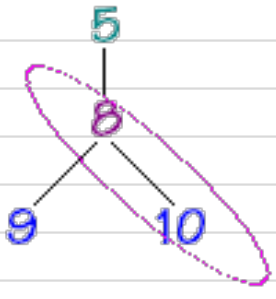


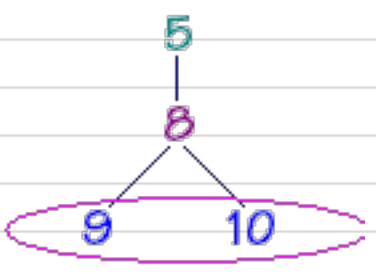

## Les différents liens entre les noeuds

Il y en a 4 :

1. Frère
2. Parent
3. Enfant
4. Ancêtre



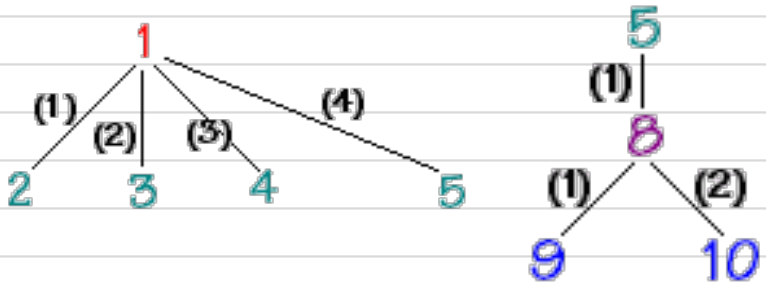




Ici :
9 est l'enfant de 8
8 est donc le parent de 9 mais aussi de 10
9 et 10 sont frères
Et 5 est l'ancêtre de 9 et 10

# Degré d'un noeud, Hauteur/profondeur d'un arbre, Degré d'un arbre, Taille d'un arbre

## Degré d'un noeud

Le degré d'un noeud est tout simple le nombre de ses descendants ce qui veut dire ses enfants.
❗ REMARQUE ❗ : lorsqu'un arbre a tous ses noeuds avec un degré de 1, on le nomme arbre dégénéré.
Ce qui en fait une liste.

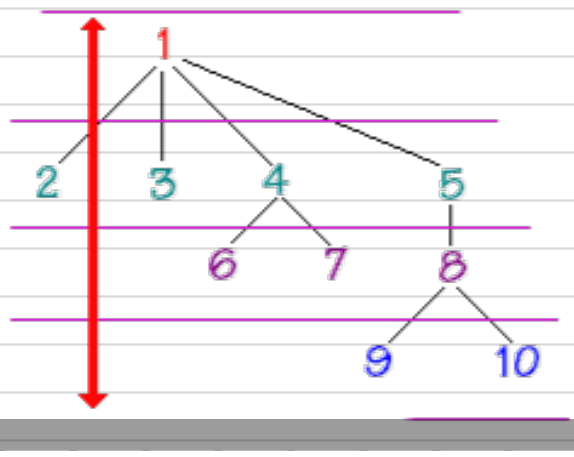



Ici sur la première image :
Le noeud 1 est de degré 4 car il a 4 enfants.
Sur la deuxième image :
Le noeud 5 n'ayant qu'un enfant est de degré 1.
Le noeud 8 est de degré 2 car il a 2 enfants.

## Hauteur / Profondeur d'un arbre

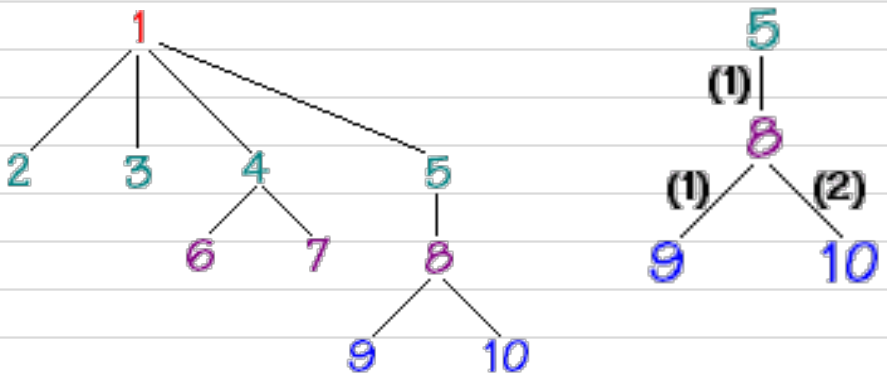
Il s'agit du nombre total de noeuds du chemin le plus long dans l'arbre.

Ici la hauteur de l'arbre est de 4.




## Degré d'un arbre

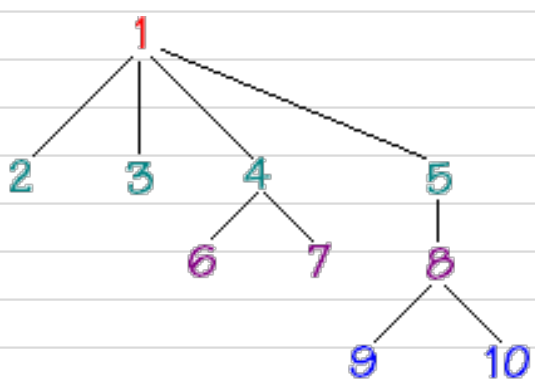
Le degré d'un arbre est tout simple le plus grand des degrés de ses noeuds.




Ici sur la première image le degré de l'arbre est de 4.
Alors que sur la deuxième image le degré de l'arbre est de 2

## Taille d'un arbre

Il s'agit du nombre total de noeuds présent dans un arbre.




Ici le nombre total de noeuds dans l'arbre est de 10.
Ce qui signifie que sa taille est de 10

# ARBRE BINAIRE EN C

## Type structuré d'un arbre :

```c
//Type strucure d'un arbre
typedef struct Arbre
{
    int valeur;
    struct Arbre *filsgauche;
    struct Arbre *filsdroit;
    struct Arbre *parent;
}Arbre;
```

## Fonction pour créer un noeud avec juste sa valeur

```c
Arbre *creaNoeud (int nX)
/*fonction de création d'un noeud en entrant seulement la valeur de celui-ci*/
{
    Arbre *aArbreT = malloc(sizeof(*aArbreT));
    if (aArbreT == NULL)
    {
        printf("Erreur\n");
        return NULL;
    }

    aArbreT->valeur = nX;
    aArbreT->filsgauche = NULL;
    aArbreT->filsdroit = NULL;
    aArbreT->parent = NULL;

    //j'indique à chaque fois le noeud cree pour etre sur que l'action a ete effectue
    printf("creation %d\n",aArbreT->valeur);

    return aArbreT;
}
```

## Fonction pour assembler les noeuds

```c
Arbre *assemblage (Arbre *aGauche,Arbre *aDroit, int nValeur)
//Assemblage des differents arbres entre eux
{
    //On appelle la fonction creaArbre pour cree un noeud avec seulement une valeur
    Arbre *aArbreT = creaNoeud(nValeur);

    //Ensuite on va pouvoir definir le fils gauche et droit grace au parametre demander
    aArbreT->filsgauche = aGauche;
    aArbreT->filsdroit = aDroit;

    //On teste ensuite si les parametres aGauche et aDroit sont non NULL alors aArbreT est leur parent
    if (aGauche != NULL)
    {
        aGauche->parent = aArbreT;
    }
    if (aDroit != NULL)
    {
        aDroit->parent = aArbreT;
    }
    return aArbreT;
}
```

# Procédure permettant d'afficher l'arbre

```c
void affichageArbre (Arbre *aArbre)
//Affichage de l'arbre final
{
    //Si l'arbre est vide alors on ne fait rien
    if (aArbre == NULL)
    {
        return;
    }

    //Si le parent du noeud est non NULL alors on affiche la valeur du parent puis la valeur du noeud actuel
    if (aArbre->parent != NULL)
    {
        printf("[%d]\n",aArbre->parent->valeur);
        printf(" | \n");
        printf("[%d]\n",aArbre->valeur);
    }
    else
    {
        printf("(%d)------pas de parent\n",aArbre->valeur); //Ici on affiche la racine
    }

    //Affichage récursif
    if (aArbre->filsgauche != NULL)
    {
        affichageArbre(aArbre->filsgauche);
    }
    if (aArbre->filsdroit != NULL)
    {
        affichageArbre(aArbre->filsdroit);
    }

}
```

# Procédure permettant de supprimer les espaces alloués aux noeuds

```c
void nettoyer(Arbre *aArbre)
//Cette procedure permet de nettoyer les espaces alloues au different noeuds de l'arbre
{
    //Si l'arbre est vide alors on on ne fait rien
    if (aArbre == NULL)
    {
        return;
    }

    //j'indique à chaque fois le noeud supprime pour etre sur que l'action a ete effectue
    printf("suppression %d\n",aArbre->valeur);

    //On rappelle la fonction pour le fils gauche et droit pour pouvoir nettoyer tout l'arbre
    nettoyer(aArbre->filsgauche);
    nettoyer(aArbre->filsdroit);
    //On nettoie le noeud
    free(aArbre);

}
```

# Programme principal

```c
int main()
{
    /*Creation d'un arbre binaire :
                1
               /  \
              2    3
             / \  / \
            4  5  6  7
                     \
                      8
    */

    Arbre *aArbre = assemblage(
        assemblage(creaNoeud(4),creaNoeud(5),2),
            assemblage(creaArbre(6),assemblage(NULL,creaNoeud(8),7),3),
            1);


    //affichage de l'arbre en entier
    affichageArbre(aArbre);
    //On nettoie les espaces alloués
    nettoyer(aArbre);

    return 0;
}
```