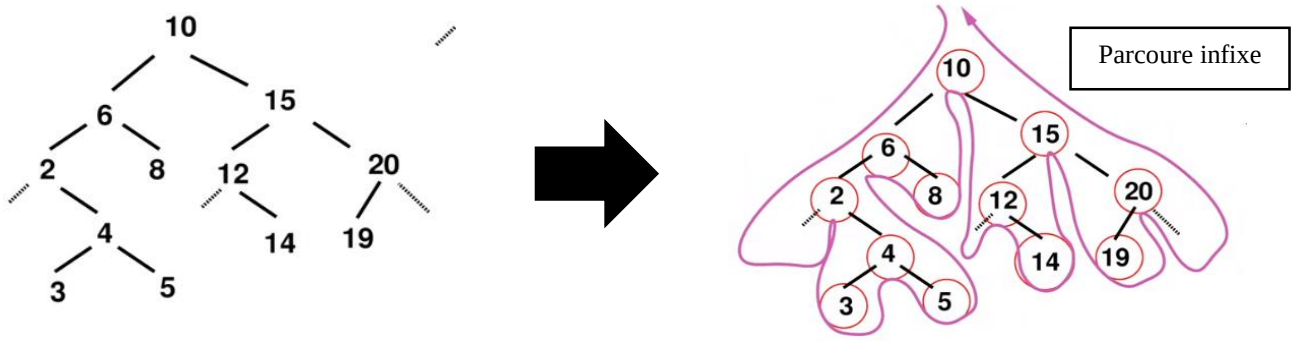
TP : Arbre binaire de recherche - Programmation récursive

Définition : Un arbre binaire de recherche (ou ABR) est un arbre binaire tel que :

- L'étiquette de la racine est
  - Supérieure à toutes les étiquettes du sous-arbre gauche
  - Inférieure à toutes les étiquettes du sous-arbre droit
- Les sous-arbres gauche et droit de la racine sont aussi des arbres binaires de recherche

On notera le type **ArbreBinRecherche** qui est **ArbreBinaire** [Nombre] avec la proprite additionnelle.

Exemple :



Travail demandé :

Si on parcourt Arbre binaire de recherche par un parcoure infixe, on obtient la liste ordonnée des éléments :

Liste infixe : 2 3 5 6 8 10 12 14 15 19 20

→ Écrire le code pour fabriquer cette liste infixe :

Le schéma récursif :

1. Arbre binaire est vide ou non vide ?
2. Si arbre binaire est vide → le résultat est la liste vide
3. Si l'arbre binaire n'est pas vide → On a un fis gauche et un fis droit et une étiquette
4. Concaténons la liste infixe de toutes les étiquettes qui se trouvent dans le sous arbre de gauche, puis placer l'étiquettes de nœuds où je suis et puis placer la liste des étiquettes de droites.

```
(liste-infixe B)
   (if (ab-noeud? B)
       (append (liste-infixe (ab-gauche B))
               (list (ab-etiquette B))
               (liste-infixe (ab-droit B)))
       (list)))
```

```c
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node* left;
    struct node* right;
};
struct node* createNode(value){
    struct node* newNode = malloc(sizeof(struct node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
struct node* insertLeft(struct node *root, int value) {
    root->left = createNode(value);
    return root->left;
}
struct node* insertRight(struct node *root, int value){
    root->right = createNode(value);
    return root->right;
}
int main(){
    struct node *root = createNode(1);
    insertLeft(root, 2);
    insertRight(root, 3);

    printf("The elements of tree are %d %d %d", root->data, root->left->data, root->right->data);
}
```