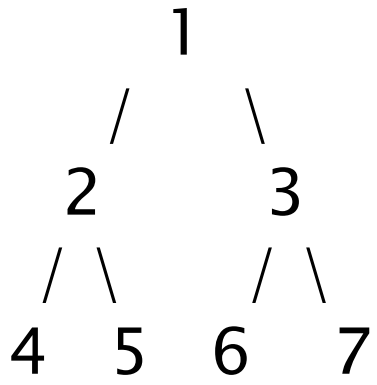
Arbres en Prolog

- Un arbre binaire est une structure pouvant contenir des données. Chaque élément de l'arbre contient une donnée et a au plus un 'parent' et deux 'enfants'.



Arbres en Prolog

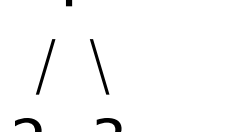
On peut representer les arbres avec une structure

t(elem, gauche, droit)

ou 'elem' est la valeur de la racine,

et gauche et droit sont les sous-arbres à gauche/droite de la racine. Un arbre vide sera represente par 'nul'.

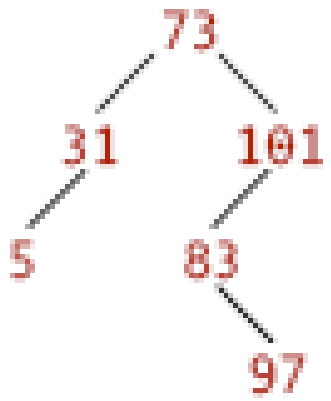
Ainsi, un arbre à un seul élément 1: t(1,nul,nul):



t(1,t(2,nul,nul),t(3,nul,nul)).

Un arbre binaire

```
t(73,
  t(31,
     t(5, nul, nul),
     nul),
  t(101,
     t(83,
        nul,
        t(97, nul, nul)),
     nul))
```



Parcours inordre

```
printInfo(nul).
printInfo(t(RootInfo,LeftSubtr,RightSubtr)) :-
      printInfo(LeftSubtr),
      write(RootInfo),
      write(' '),
      printInfo(RightSubtr).
```

```
?- printInfo(t(73, t(31, t(5, nul,
nul), nul), t(101, t(83, nul, t(97,
nul, nul)), nul))).
5 31 73 83 97 101
Yes
```

Recherche dans un arbre

```
search(Info, t(Info, _, _)).
search(Info, t(RootInfo, Left, _Right)) :-
        precedes(Info, RootInfo),
        search(Info, Left).
search(Info, t(RootInfo, _Left, Right)) :-
        precedes(RootInfo, Info),
        search(Info, Right).

precedes(Info1, Info2) :-
        Info1 < Info2 .
```

Insertion dans un arbre

```
insert(Info, nul, t(Info, nul, nul)).
insert(Info, t(RootInfo, Left, Right),
                    t(RootInfo, LeftPlus, Right)) :-
    precedes(Info, RootInfo),
    insert(Info, Left, LeftPlus).
insert(Info, t(RootInfo, Left, Right),
                    t(RootInfo, Left, RightPlus)) :-
    precedes(RootInfo, Info),
    insert(Info, Right, RightPlus).
```

Retrait à la racine d'un arbre

```
delete(Info, t(Info, nul, Right), Right).
delete(Info, t(Info, Left, nul), Left).
delete(Info,  t(Info, Left, Right),
                    t(NewRoot, NewLeft, Right)) :-
    removeMax(Left, NewLeft, NewRoot).

% removeMax(Tree,NewTree,Max)
```

Retrait dans un arbre

```
delete(Info, t(RootInfo, Left, Right),
                    t(RootInfo, LeftSmaller, Right)) :-
          precedes(Info, RootInfo),
          delete(Info, Left, LeftSmaller).
delete(Info, t(RootInfo, Left, Right),
                    t(RootInfo, Left, RightSmaller)) :-
          precedes(RootInfo, Info),
         delete(Info, Right, RightSmaller).
```

Retrait du max élément

```
removeMax(t(Max, Left, nul), Left, Max).
removeMax(t(Root, Left, Right),
                    t(Root, Left, RightSmaller), Max) :-
        removeMax(Right, RightSmaller, Max).
```