Prolog : TP 2

Listes, arbres et ordre supérieur

1. Écrire le prédicat `a_plat/2` qui « met à plat » une liste de listes imbriquées (à un niveau quelconque, on rappelle que le langage n'est pas typé).

```
?- a_plat([[[1], [3,4]], 1, [1,2,3]], L).
L = [1, 3, 4, 1, 1, 2, 3]
```

2. Écrire un prédicat qui effectue un parcours en largeur d'abord d'un arbre binaire codé avec les symboles `n/2` (nœud) et `f/1` (feuille). Indication : utiliser une file.

```
?- largeur(n(n(f(4),f(5)),f(6)), L).
L = [6, 4, 5]
```

3. Le prédicat binaire infixe `=..` permet de décomposer ou de construire un terme : `Term =.. [F | Args]` réussit si F est le constructeur du terme `Term` et `Args` ses arguments. Par exemple :

```
?- f(1, f(2)) =.. [F | As].
F = f
As = [1, f(2)]
```

```
?- G =.. [append, [1,2], [3], [1,2,3]].
G = append([1, 2], [3], [1, 2, 3])
```

   Écrire un prédicat `symbols /2` qui calcule la liste des symboles d'un terme.

```
?- symbols(f(f(1), g(2)), L).
L = [f, 1, g, 2]
```

4. Le prédicat prédéfini `call/1` permet d'évaluer un terme comme s'il était un but. Par exemple

```
?- But = (X = 1), call(But).
X = 1
But = (1 = 1)
```

   Écrire le prédicat `map /3` qui applique un prédicat à chacun des éléments d'une liste.

```
?- map(reverse, [[1,2],[3],[5,6,7]], L).
L = [[2, 1], [3], [7, 6, 5]]
```