

**Nombre d'esquerra i dreta en un arbre binari**

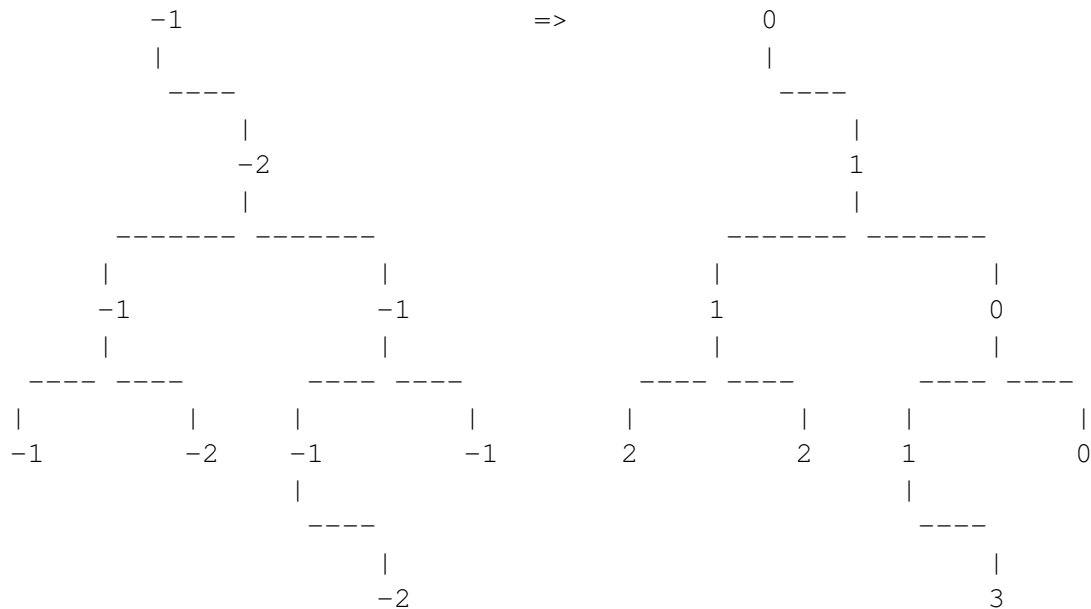
**X86374\_ca**

Implementeu una funció **RECURSIVA** que, donat un arbre binari d'enters amb valors -1 i -2 als nodes, retorna un nou arbre amb la mateixa estructura que l'inicial, i a on per a cada posició p hi ha un valor d'acord al següent criteri. Si l'arbre inicial tenia -1 a posició p, llavors el nou arbre té a posició p el nombre de cops que s'escull fill-esquerra en el camí des de l'arrel fins a posició p. En canvi, si l'arbre inicial tenia -2 a posició p, llavors el nou arbre té a posició p el nombre de cops que s'escull fill-dret en el camí des de l'arrel fins a posició p.

```
// Pre: Sigui T el valor inicial de l'arbre t que es rep com a paràmetre.
//      Els valors guardats a T son tots -1 o -2.
// Post: Sigui T' l'arbre retornat. T i T' tenen exactament la mateixa estructura
//       Sigui p una posició qualsevol de T'.
//       Si T té un valor -1 a posició p, llavors T' té el nombre de cops que
//       s'escull fill-esquerra en el camí des de l'arrel fins a posició p.
//       Si T té un valor -2 a posició p, llavors T' té el nombre de cops que
//       s'escull fill-dret en el camí des de l'arrel fins a posició p.
BinTree<int> numLeftRight(BinTree<int> t);
```

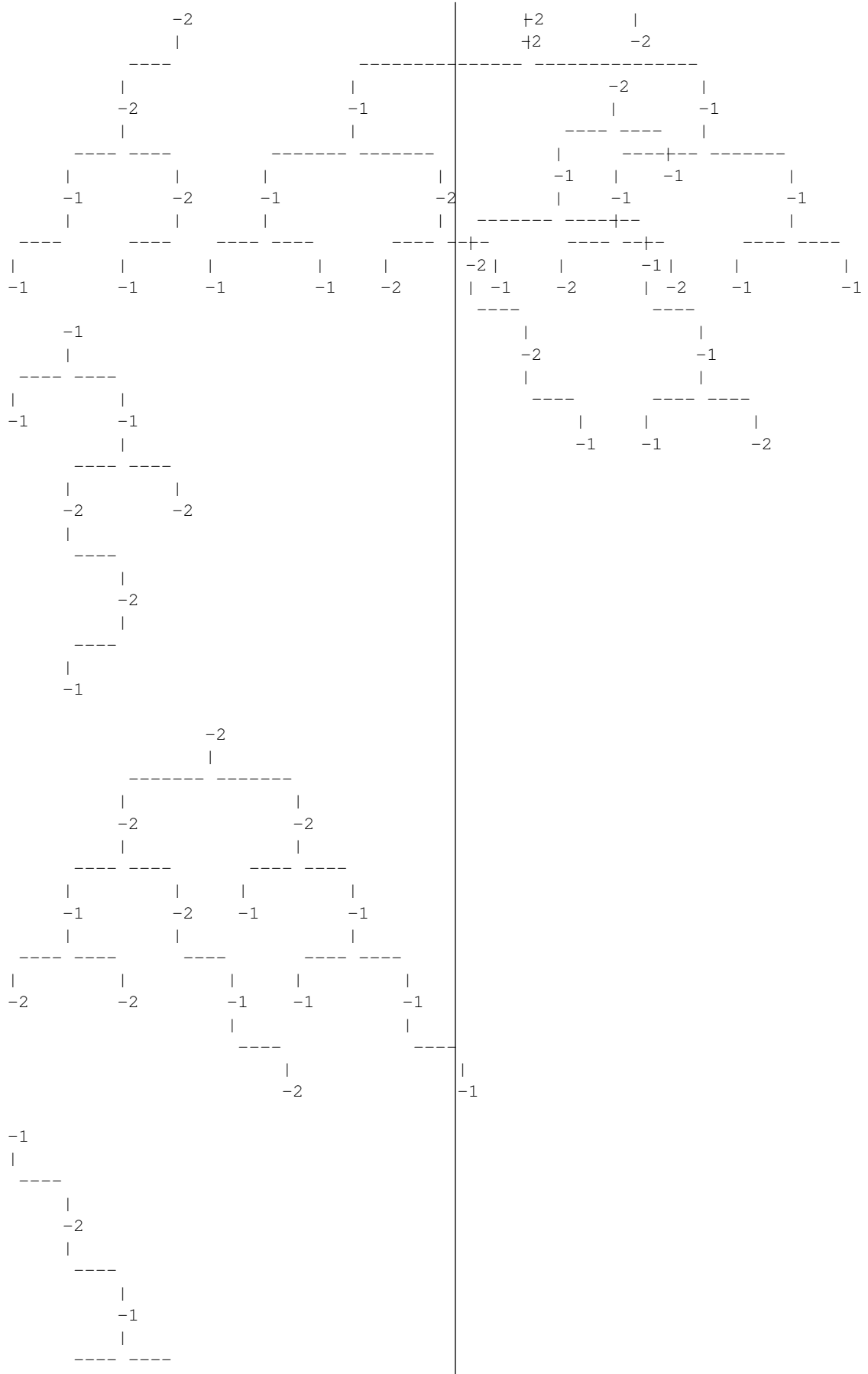
Aquí tenim un exemple de comportament de la funció:

```
numLeftRight(-1(-2(-1(-1,-2),-1(-1(-,-2),-1)))) = 0(1(1(2,2),0(1(,3),0)))
```

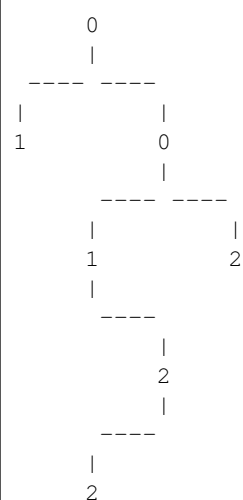
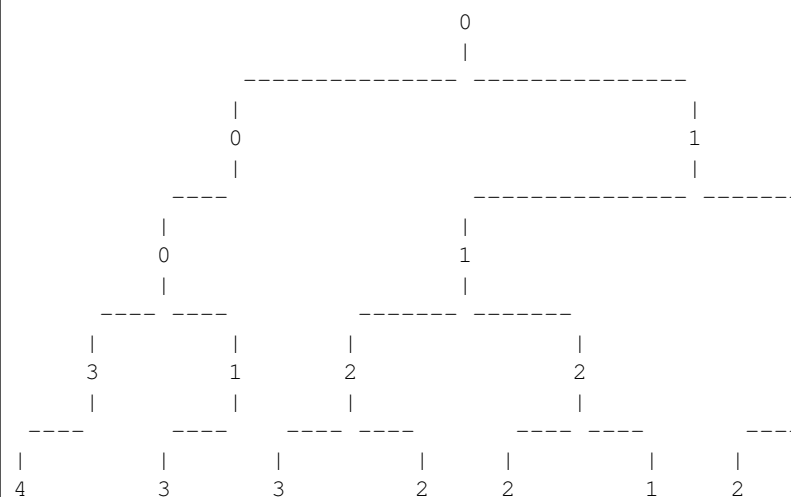
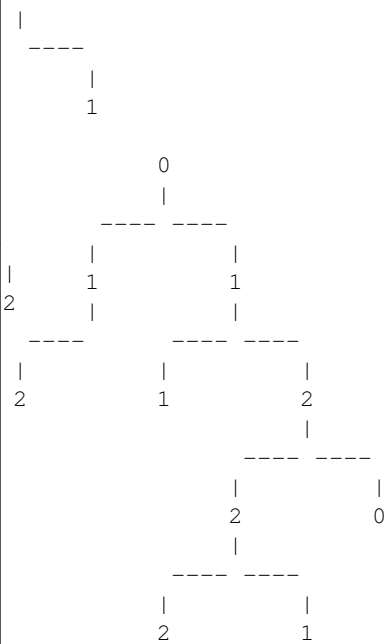
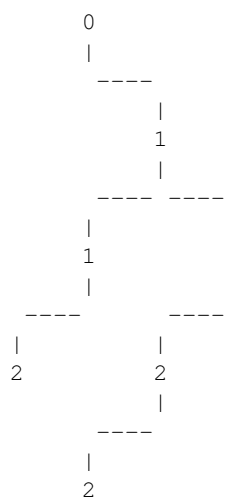
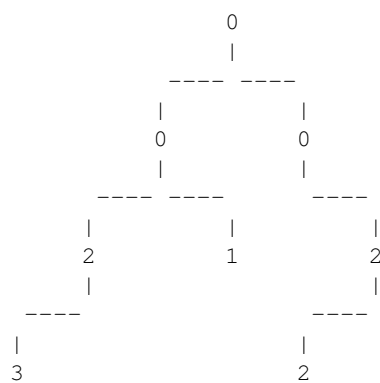
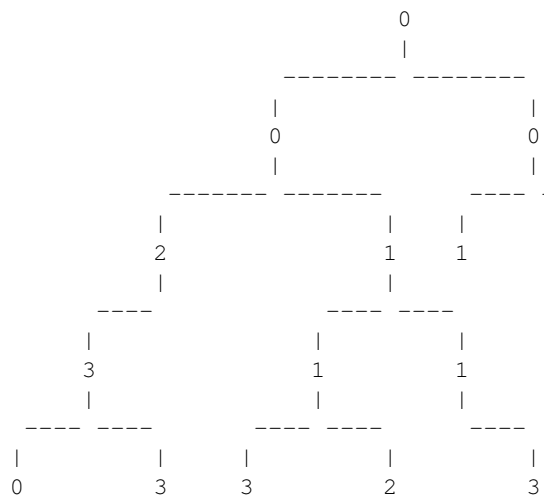


Fixeu-vos que l'enunciat d'aquest exercici ja ofereix uns fitxers que haureu d'utilitzar per a compilar: main.cc, BinTree.hh, numLeftRight.hh. Només cal que creeu numLeftRight.cc, posant-hi els includes que calguin i implementant la funció numLeftRight. Només cal que pugueu numLeftRight.cc al jutge.





# Exemple de sortida 1



0  
|

