

---

## Arbre màxim

X41728\_ca

---

Implementeu una funció **RECURSIVA** que, donats dos arbres binaris d'enters positius, obté un nou arbre que conté, per a cada posició, el màxim dels valors dels dos arbres de partida en les mateixes corresponents posicions. En cas que un dels arbres no tingui un valor definit en una posició, s'agafa el valor de l'altre arbre. Aquesta és la capçalera:

```
// Pre: Rep dos arbres binaris d'enters positius a1 i a2.
// Post: Retorna un arbre, on a la seva arrel hi ha el màxim de les arrels de
// en l'arrel del fill esquerra, el màxim de les arrels dels fills esquerra de
// i així successivament.
// Quan un dels arbres no té valors definits en alguna posició, l'arbre resulta
// el valor de l'altre arbre en aquella posició.

arbreBin<int> maximumTree(arbreBin<int> a1, arbreBin<int> a2);
```

Aquí tenim un exemple de comportament de la funció, a on es mostren els elements dels arbres com `valor_arrel(arbre_fill_esquerra, arbre_fill_dret)`. Aquesta notació és expressament ambigua. Per exemple, els strings `33(45(,), 51(,))` i `33(45, 51)` representen el mateix arbre.

```
8(8(, 5), 8(2, 8))
9(7(9(, ,))
=>
9(8(9, 5), 8(2, 8))
```

Fixeu-vos que l'enunciat d'aquest exercici ja ofereix uns fitxers que haureu d'utilitzar per a compilar: `Makefile`, `program.cpp`, `arbreBin.hpp`, `maximumTree.hpp`, `maximumTree.cpp`. Només cal que creeu `maximumTree.cpp`, posant-hi els includes que calguin i implementant la funció `maximumTree`. I quan pugeu la vostra solució al jutge, només cal que pugeu un tar construït així:

```
tar cf solution.tar maximumTree.cpp
```

## Entrada

L'entrada té un nombre arbitrari de casos. Cada cas consisteix en dues línies. Cadascuna d'aquestes dues línies té un string que descriu un arbre binari. Fixeu-vos en que el programa que us oferim ja s'encarrega de llegir aquestes entrades.

## Sortida

Per a cada cas, cal escriure l'arbre binari resultant de calcular el màxim entre els dos arbres d'entrada. Fixeu-vos en que el programa que us oferim ja s'encarrega d'escriure l'arbre resultant.

### Exemple d'entrada

```
8(8(,5),8(2,8))
9(7(9,))
6(4(8,4),)
3(4,6)
9(6(5,))
7(3(1(,3),),8)
6(,7)
7(,9)
1(4(,3(3,1)),2(4(,1),))
2(5,5(6,1(1,7)))
2(,3(6(3,3),4(5,1)))
6(7(,9),5(7,))
3(9,2(,3))
6(7(8,3),)
8(3(3(8,)),2(3,2)),4(3,1)
3(7,9(5,4))
8(,4)
9(7,)
7(4,)
2(2,7)
```

### Exemple de sortida

```
9(8(9,5),8(2,8))
6(4(8,4),6)
9(6(5(,3),),8)
7(,9)
2(5(,3(3,1)),5(6(,1),1(1,7)))
6(7(,9),5(7(3,3),4(5,1)))
6(9(8,3),2(,3))
8(7(3(8,)),2(3,2)),9(5,4)
9(7,4)
7(4,7)
```

### Observació

La vostra funció i subfuncions que creu han de treballar només amb arbres. Heu de trobar una solució **RECURSIVA** del problema. En les crides recursives, incloeu la hipòtesi d'inducció, és a dir una explicació del que es compleix després de la crida, i també la funció de fita/decreixement o una justificació de perquè la funció recursiva acaba.

### Informació del problema

Autor :

Generació : 2022-02-09 15:56:01

© *Jutge.org*, 2006–2022.

<https://jutge.org>