Adaptar BinaryTree per a mantenir informació sobre la sumaX27581_ca

L'objectiu d'aquest exercici és afegir un nou mètode `getSum` a la classe Genèrica `BinaryTree` que retorni la suma dels nodes de l'arbre. Obviament, això només tindrà sentit per a tipus de dades per als quals la operació de suma (+) està definida. Assumirem com a precondició que mai es crida a aquesta funció amb l'arbre buit. En tal cas, una opció raonable seria emetre un missatge d'error i abortar l'execució. En el cas d'arbres amb un sol node, la suma serà el propi node.
Una opció seria que aquest mètode calculés la suma dels nodes, per exemple recursivament, i la retornés, però aquest enfoc seria massa lent per a poder superar els jocs de proves privats. Aquesta operació hauria de tenir cost constant, i per això, convindrà afegir informació adicional a la classe que permeti mantenir actualitzada informació sobre la suma. A continuació donem una guia de com fer això.
D'entre els fitxers que s'adjunten en aquest exercici, trobareu `BinaryTree.old.hpp`, a on hi ha una implementació de la classe genèrica `BinaryTree`. En primer lloc, haureu de fer:

```
cp BinaryTree.old.hpp BinaryTree.hpp
```

A continuació, heu de fer tot un seguit de canvis sobre la classe `BinaryTree` definida a `BinaryTree.hpp`:

- Heu d'afegir un nou atribut `T sum`.
- Heu d'afegir un nou mètode privat per a actualitzar la suma de l'arbre i la suma dels seus antecessors (els arbres que tenen a l'arbre actual com a subarbre). Això es pot fer de forma recursiva o iterativa. Una possible manera iterativa és:

  ```
  void updateSum()
  {
  if (isEmpty()) {
  cerr << "Error: sum on empty tree" << endl;
  exit(1);
  }
  BinaryTree<T> *pt = this;
  while (pt != NULL) {
  pt->sum = ...;
  if (not pt->left->isEmpty())
  pt->sum += ...;
  if (not pt->right->isEmpty())
  pt->sum += ...;
  pt = pt->parent;
  }
  }
  ```

  Una possible manera recursiva és:

```
void updateSum()
{
if (isEmpty()) {
cerr << "Error: sum on empty tree" << endl;
exit(1);
}
sum = ...;
if (not left->isEmpty())
sum += ...;
if (not right->isEmpty())
sum += ...;
if (parent != NULL) parent->updateSum();
}
```

- A les constructores i a l'operació d'assignació heu d'afegir crides a `updateSum`. Aquí heu de vigilar: si l'arbre creat és buit, no haurieu de cridar a `updateSum` de l'arbre implícit, però potser sí a `updateSum` del nostre antecessor en casos especials de l'operació assignació per als quals tinguem un antecessor prèviament definit.
- Heu d'implementar el mètode `getSum`, simplement retornant el valor del nou atribut.

D'entre els fitxers que s'adjunten a l'exercici també hi ha `program.cpp` (programa principal) i `Makefile` per a compilar. Per a pujar la vostra solució, heu de crear el fitxer `solution.tar` així:

```
tar cf solution.tar BinaryTree.hpp
```

Entrada

El programa principal té una variable d'arbre d'enters `t`, inicialment buida, i llegeix instruccions que, o bé mostren com és `t`, o bé modifiquen algun subarbre de `t` o mostren la suma d'algun subarbre de `t`. Les instruccions que mostren `t` són simplement de la forma `<< t`. Les altres instruccions comencen per `t`, seguit d'una seqüència de `.left` o `.right`. Finalment, o bé la instrucció acaba amb `.sum`, cas en el qual s'escriurà la suma del corresponent subarbre, o ve seguida de `= t'`, on `t'` és un string que representa un arbre, cas en el qual `t'` (com a arbre) serà assignat al corresponent subarbre de `t`. Per exemple:

```
t = 3(4,5(1,2))
<< t
t.sum
t.left.sum
t.right.sum
t.right.left = 8(9,10)
<< t
t.right.sum
```

La sortida de la seqüència anterior és:

```
3(4,5(1,2))
15
4
```

```
8
3(4,5(8(9,10),2))
34
```

Com podeu observar, el `sum` d'un arbre que està per sobre del que hem assignat també ha estat actualitzat.
Se suposa que la seqüència d'entrada serà correcta (sense accessos fora de l'arbre, tot i que sí que es pot accedir a subarbres buits de l'arbre).
El programa principal que us oferim ja s'encarrega de llegir aquestes entrades i fer les crides als corresponents mètodes de la classe BinaryTree. Només cal que feu les modificacions abans esmentades dins el fitxer `BinaryTree.hpp`.

Sortida

Per a cada instrucció `<< t`, s'escriurà el contingut actual de l'arbre. Per a cada instrucció acabada en `sum`, s'escriurà la suma del subarbre indicat. El programa que us oferim ja fa això. Només cal que feu les modificacions abans esmentades dins el fitxer `BinaryTree.hpp`.

Exemple d'entrada 1

```
t = 7(2,5)
t.sum
<< t
t = 5(,1)
t.sum
<< t
t.left = 4(2(,3),2)
t.left.sum
<< t
t.left.left = 7(3,)
t.left.sum
<< t
t.right = 5(6(1,),2)
t.sum
<< t
t.left.right = 5(,8(,3))
t.sum
<< t
t.left.right.left = 1(3,4(3,2))
t.sum
<< t
t.right.right = 2(5(2,2),)
t.right.right.sum
<< t
t.left.left = 6
t.left.left.sum
<< t
t.right.right.left = 1
t.right.sum
<< t
```

Exemple de sortida 1

```
14
7(2,5)
6
5(,1)
11
5(4(2(,3),2),1)
16
5(4(7(3,),2),1)
35
5(4(7(3,),2),5(6(1,),2))
49
5(4(7(3,),5(,8(,3))),5(6(1,),2))
62
5(4(7(3,),5(1(3,4(3,2)),8(,3))),5(6(1,),2))
11
5(4(7(3,),5(1(3,4(3,2)),8(,3))),5(6(1,),2(5(2,2),)))
6
5(4(6,5(1(3,4(3,2)),8(,3))),5(6(1,),2(5(2,2),)))
15
5(4(6,5(1(3,4(3,2)),8(,3))),5(6(1,),2(1,)))
```

Exemple d'entrada 2

```
t = 7
<< t
t = 5
<< t
t = 4(5,1)
<< t
t.left.left = 4(2,3)
t.left.sum
<< t
t.right = ()
```

```
t.sum
<< t
t.left = 4(7,)
t.sum
<< t
t.left.right = 2
t.sum
<< t
t = 2
<< t
t = ()
<< t
t = ()
<< t
```

Exemple de sortida 2

```
7
5
4(5,1)
14
4(5(4(2,3),),1)
18
4(5(4(2,3),),)
15
4(4(7,),)
17
4(4(7,2),)
2
()
()
```

Informació del problema

Autoria: PRO1

Generació: 2026-01-25T21:06:31.639Z