

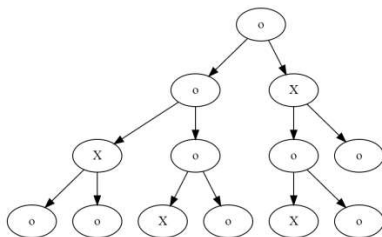
## Alpinismo

El objetivo de este control es familiarizarse con el procesamiento de árboles binarios.

## 1) El problema

Podemos utilizar los árboles binarios para representar los caminos en la falda de una montaña. La raíz del árbol representa la cima, de la que salen uno o dos caminos. Cada camino finaliza en una *intersección* distinta. Dicha intersección puede ser, bien una *meta*, bien una *bifurcación*. De las bifurcaciones parten, a su vez, otros dos nuevos caminos que nunca se volverán a conectar. Un escalador está en la cima de la montaña (raíz del árbol) y se da cuenta de que en distintas intersecciones (marcadas en el árbol con 'X') hay amigos que necesitan su ayuda para subir. Tiene que bajar a cada una de las 'X' y ayudarles a subir de uno en uno. Para recorrer cada tramo del camino (tramo = distancia entre intersecciones), el escalador necesita una hora en cada uno de los dos sentidos (es decir, una hora en bajar, y otra hora en subir).

El problema consiste en determinar cuánto tiempo tarda el escalador en ayudar a sus amigos. Por ejemplo, dada la falda de montaña representada por el siguiente árbol:



el escalador necesitará 18 horas en ayudar a todos sus amigos.

## 2) Trabajo a realizar

Se debe construir un programa que lea una serie de filas, cada una representando una falda de montaña del tipo descrito anteriormente, e imprima por la salida el tiempo que necesita el escalador en ayudar a sus amigos.

Los árboles se codifican en la entrada de acuerdo con el siguiente criterio:

- El árbol vacío se representa como #
- Un árbol simple con raíz  $A$  se representa como  $[A]$
- Un árbol compuesto con raíz  $A$  se representa como  $(\tau_i \ A \ \tau_d)$ , donde  $\tau_i$  es la representación del hijo izquierdo, y  $\tau_d$  la del derecho.

De esta forma, el árbol mostrado anteriormente se codificará como:

$$((([o]X[o]) \circ ([X] \circ [o])) \circ (([X] \circ [o])X[o]))$$

**Ejemplo de entrada / salida:**

Entrada	Salida
$((([X] \circ [X]) \circ ([X] \circ [X])) \circ (([X] \circ [X]) \circ [X]))$	18
$((([X] \circ [X]) \circ ([X] \circ [X])) \circ (([X] \circ [X]) \circ [X]))$	16
$((([X] \circ [X]) \circ ([X] \circ [X])) \circ (([X] \circ [X]) \circ [X]))$	12
$((([X] \circ [X]) \circ ([X] \circ [X])) \circ (([X] \circ [X]) \circ [X]))$	0

Se proporciona el archivo `main.cpp` en el que se implementa la lógica de entrada / salida necesaria. El código proporcionado no debe modificarse.

Hay que añadir a dicho archivo la implementación de la siguiente función:

```
// Devuelve el tiempo que necesita el escalador para ayudar a sus amigos.
// Parámetros:
//     falda: La representación como árbol binario de los caminos de
//            la falda de la montaña, indicando la situación de amigos
//            en las intersecciones (intersecciones marcadas con X).
// Resultado:
//     Tiempo que necesita el escalador para ayudar a sus amigos
// Precondición: En 'falda' hay una representación válida de
//               rutas y amigos (esta precondición no es
//               necesario comprobarla)
int tiempoAyuda(const Arbin<char>& falda);
```

Aparte de esta función, podrán añadirse todas aquellas funciones auxiliares que se consideren necesarias.