1. Escriba una función que busque en un árbol binario de nodos con contenidos **`const char *`**. La función solicitada, **`find(word)`**, debe retornar un puntero al nodo buscado. Use excepciones para indicar el caso de "no encontrado". ?
2. Defina una clase **`Int`** que se comporte como el tipo nativo **`int`**, excepto que dispara excepciones en caso de over o underflow.
3. Tome las operaciones básicas para abrir, cerrar, leer y escribir en archivos utilizando las interfaces C y provea funciones equivalentes que llaman a las funciones en C, pero que disparan excepciones en caso de errores.
4. Esciba un template completo **`Vector`** con excepciones de **`Range`** y **`Size`**.
5. Escriba una clase **`Lock`** para encapsular algún mecanismo de concurrencia de un sistema.
6. Suponga un programa que dispare una excepción y que el handler apropiado comienza a ejecutarse. Ahora suponga que el handler mismo dispara nuevamente la excepción. Crea esto una condición de recursión infinita? Escriba un programa para comprobar la respuesta. se ejecuta terminate
7. Utilice herencia para crear varias clases derivadas de **`runtime_error`**. Muestre que puede escribir un catch en base a la clase base y atrapar excepciones de las clases derivadas. ??
8. Escriba un programa que ilustre que todos los destructores para los objetos construidos en un bloque son llamados si se dispara una excepción dentro del bloque.
9. Escriba un programa que ilustre que una función con su propio bloque **`try`** no tiene que atrapar todo posible error generado dentro del **`try`** y que algunas excepciones pueden ser manipuladas en scopes externos.