Práctica 2

1. Lea la implementación provista de árboles binarios enlazados[1], y el ejemplo presentado en el archivo `main.c`. Asegúrese comprenderlo.

2. En cada caso escriba una función que:

a) Calcule la suma de los elementos de un árbol de enteros.

b) Cuente la cantidad de nodos de un árbol de enteros.

c) Calcule la altura de un árbol de enteros.

3. Describa en que orden **btree_foreach** recorre el árbol según lo visto en clase. Agregue como parámetro un elemento del siguiente tipo a la función, e implemente los recorridos correspondientes:

```
typedef enum {
  BTREE_TRAVERSAL_ORDER_IN,
  BTREE_TRAVERSAL_ORDER_PRE,
  BTREE_TRAVERSAL_ORDER_POST
} BTreeTraversalOrder;
```

4. Escriba una función que calcule el mínimo elemento de un árbol, y reemplace todos los nodos con este valor. ¿Puede proponer una versión que recorra el árbol solo una vez? Si reemplazamos el dato entero por un puntero a un entero, ¿es posible? De serlo, escriba esta versión.

5. Escribir un tipo de datos `BSTree` que implemente árboles binarios ordenados (enlazados). Diseñe y programe las siguientes funciones:

- **bstree_insert** que agregue un elemento a un árbol binario ordenado.
- **bstree_contains** que diga si un elemento está en un árbol binario ordenado.
- **bstree_nelements** que devuelva la cantidad de elementos de un árbol binario ordenado.
- **bstree_height** que devuelva la altura de un árbol binario ordenado.
- **bstree_foreach** que aplique una función en cada nodo de un árbol binario ordenado.

Indique qué condición es necesaria para que tras agregar varios elementos, el árbol resultante tenga en realidad forma de lista. Haga un análisis de la cantidad de nodos que hay que visitar en el caso de buscar un elemento en esa lista, y compárelo con un árbol que no tenga la forma de lista.

6. Escribir una función `mirror` que dado un árbol binario, genere el árbol binario espejo: el hijo derecho de cada nodo pasa a ser izquierdo y el izquierdo pasa a ser derecho.

7. Un "árbol rosa" (rose tree) es una forma de árbol general compuesto de nodos, donde cada nodo tiene un dato y una lista de nodos (sus hijos).
Utilizando las listas generales (`void *`), implemente los árboles rosa.

8. Implemente árboles binarios completos utilizando arreglos. Procure que la interfaz provea las siguientes funciones:

- **cbtree_new** que crea un nuevo árbol de una cantidad de niveles dado.
- **cbtree_destroy** que recibe un árbol y lo destruye.

[1] `http://dcc.fceia.unr.edu.ar/~erivas/estructuras/src/arboles/`

- **cbtree_insert** que agregue un elemento a un árbol en la siguiente posición disponible.
- **cbtree_nelements** que devuelve la cantidad de elementos en el árbol.
- **cbtree_foreach** que aplica una función a cada nodo del árbol.

9. Un árbol de Fibonacci es un árbol donde cada nodo contiene de dato un número natural, y como hijos cada uno con los números de Fibonacci del que depende.

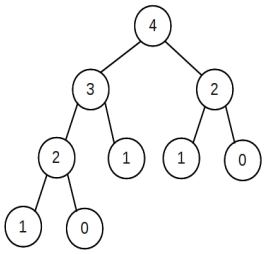

Figura 1: Ejemplo. Este es el árbol de Fibonacci correspondiente al 4.

a) Escribir una función que dado un natural, genere el árbol de Fibonacci correspondiente a ese natural.

b) Escribir una función que dado un árbol binario, calcule la cantidad de hijos que tiene.

c) Proponga una implementación (ineficiente) de la función `fib`.