**Jose Isai Rodriguez Kingsley / 100026556**

**Declaración del tipo de dato**

Ha declarado el tipo de datos representado por el árbol AVL. Esto formaliza las cosas y, a lo largo de este documento, le permite definir operaciones sobre tipos de datos abstractos.

**Algunas consideraciones al implementar tipos de datos abstractos**

Las declaraciones que se enumeran a continuación no se cotizan a su valor nominal. Cada programador tiene su propio estilo y forma de resolver el problema.

Como puede ver en la siguiente lista, la única diferencia entre un nodo en un árbol AVL y un nodo en un árbol binario regular es la variable de altura de la estructura del nodo.

Los nodos de árbol pueden almacenar cualquier tipo de datos complejos. En aras de la simplicidad, este documento utiliza el tipo de datos más simple que admite la comparación: entero (tipo int Ansi C). Si los datos almacenados en cada nodo son más complejos (como una estructura) o se almacenan dinámicamente en la memoria, entonces ciertas funciones deben optimizarse para administrarlos. Por ejemplo, debe pasar las funciones de comparación, equivalentes y de memoria libre como parámetros.

La declaración de tipo de datos abstracto para el árbol AVL se muestra a continuación:

*typedef struct AVLNode AVLTree;*

*struct AVLNode*

*{*

*int dato;* *➊*

*AVLTree izq;*

*AVLTree der;*

*int altura;* *➋*

*};*

➊ Como se mencionó anteriormente, en aras de la simplicidad, la información almacenada en cada nodo del árbol es un número entero.

➋ Cada nodo almacena su propia altura relativa a la raíz absoluta del árbol en el que está trabajando.

A continuación, declaramos las operaciones básicas del árbol binario. Úselo para acceder al árbol AVL del tipo de datos abstracto.

*/\* Constructores \*/*

*AVLTree \*vacio (void);*

*/\* devuelve un árbol AVL vacío \*/*

*AVLTree \*hacer (int x, AVLTree \* izq, AVLTree \* der);*

*/\* devuelve un nuevo árbol formado por una raíz con valor x, subárbol izquierdo el árbol izq y subárbol derecho el árbol der. \*/*

*/\* Predicados \*/*

*bool es\_vacio (AVLTree \* t);*

*/\* devuelve true sii. t es un árbol vacío. \*/*

*/\* Selectores \*/*

*AVLTree \*izquierdo (AVLTree \* t);*

*/\* devuelve el subárbol izquierdo de t. \*/*

*AVLTree \*derecho (AVLTree \* t);*

*/\* devuelve el subárbol derecho de t. \*/*

*int raiz (AVLTree \* t);*

*/\* devuelve el valor de la raíz del árbol t. Precondición: !es\_vacio(t) \*/*

*int altura (AVLTree \* t);*

*/\* devuelve la altura del nodo t en el árbol \*/*

*/\* Destructures \*/*

*void destruir (AVLTree \* t, void (\*free\_dato) (int));*

*/\* libera la memoria ocupada por los nodos del árbol. Si los datos almacenados en cada nodo están almacenados dinámicamente y se los quiere liberar también, debe pasarse como segundo parámetro una función de tipo void func(int t) que libere la memoria de objetos int. Si los datos no están almacenados dinámicamente o simplemente no se los quiere destruir (liberar de memoria), pásese como segundo parámetro NULL. Nota: Función Recursiva! \*/*