An AVL Tree Class

Version 1

The AVL tree module

Here is a conforming *avl.h* file:

```
/*** AVL binary search tree class ***/

#ifndef __AVL_INCLUDED__
#define __AVL_INCLUDED__

#include <stdio.h>

typedef struct avl AVL;

extern AVL *newAVL(
    void (*)(void *,FILE *),           //display
    int (*)(void *,void *),            //comparator
    void (*)(void *));                 //freeing function
extern void insertAVL(AVL *,void *);
extern int findAVLcount(AVL *,void *);
extern void *findAVL(AVL *,void *);
extern void *deleteAVL(AVL *,void *);
extern int sizeAVL(AVL *);
extern int duplicatesAVL(AVL *);
extern void statisticsAVL(AVL *,FILE *);
extern void displayAVL(AVL *,FILE *);
extern void displayAVLdebug(AVL *,FILE *);
extern void freeAVL(AVL *);

#endif
```

Here are some of the behaviors your methods should have. This listing is not exhaustive; you are expected, as a computer scientist, to complete the implementation in the best possible and most logical manner.

- *newAVL* - The constructor is passed three functions, one that knows how to display the generic value to be stored, one that can compare two of these generic values, and one that can free generic values. The AVL object will store a pointer to a BST object, constructed with AVL-specific display, comparator, and freeing functions.
- *findAVLcount* - This method returns the frequency of the searched-for value. If the value is not in the tree, the method should return zero.
- *findAVL* - This method returns the searched-for value. If the value is not in the tree, the method should return NULL.
- *deleteAVL* - The method starts by finding the generic value stored in the tree that matches the given value. If the frequency count of the stored value is greater than one, this method reduces the frequency count and returns NULL. If the frequency count is one, however, the stored value is removed from the tree and is returned.
- *sizeAVL* - This method returns the number of nodes currently in the tree. It should run in amortized constant time.
- *duplicatesAVL* - This method returns the number of duplicate values currently in the tree. It should run in amortized constant time. This should be equal to the net number of AVL insertions minus the number of nodes in the underlying BST.
- *statisticsAVL* - This method should display the number of duplicates. Then the method calls the BST statistics method.
- *displayAVL* The method calls the tree using a level-order traversal, via the decorated display method of the underlying data structure.
- *displayAVLdebug* The method calls the display method of the underlying data structure.

Some of these methods will be wrappers for the similarly named *BST* methods, while others will add some functionality. You will need a private function to swap values. It should look something like:

```
function swapper(BSTNODE *a,BSTNODE *b)
    {
    AVLVALUE *ta = getBSTNODEvalue(a);
    AVLVALUE *tb = getBSTNODEvalue(b);

    /* swap the values stored in the AVL value objects */
```

```
        void *vtemp = ta->value;
        ta->value = tb->value;
        tb->value = vtemp;

        /* swap the counts stored in the AVL value objects */
        int ctemp = ta->count;
        ta->count = tb->count;
        tb->count = ctemp;

        /* note: AVL heights and balance factors are NOT swapped */
        }
```

The *swapper* function is passed as the third argument to the *BST* constructor.

The only local includes a *AVL* module should have are *avl.h* and *bst.h*.