A Simple Binary Search Tree Class

Version 1e

The *BST* module

This module defines a simple search tree that can be adapted for more sophisticated uses (say, a self-balancing tree) with a minimum of code rewrites.

Here is a conforming *bst.h* file:

```
    #ifndef __BST_INCLUDED__
    #define __BST_INCLUDED__

    #include <stdio.h>

    typedef struct bstnode BSTNODE;

    extern BSTNODE *newBSTNODE(void *v);
    extern void    *getBSTNODEvalue(BSTNODE *n);
    extern void    setBSTNODEvalue(BSTNODE *n,void *value);
    extern BSTNODE *getBSTNODEleft(BSTNODE *n);
    extern void    setBSTNODEleft(BSTNODE *n,BSTNODE *replacement);
    extern BSTNODE *getBSTNODEright(BSTNODE *n);
    extern void    setBSTNODEright(BSTNODE *n,BSTNODE *replacement);
    extern BSTNODE *getBSTNODEparent(BSTNODE *n);
    extern void    setBSTNODEparent(BSTNODE *n,BSTNODE *replacement);
    extern void    freeBSTNODE(BSTNODE *n,void (*freer)(void *));

    typedef struct bst BST;

    extern BST *newBST(
        void (*)(void *,FILE *),           //display
        int (*)(void *,void *),            //comparator
        void (*)(BSTNODE *,BSTNODE *),     //swapper
        void (*)(void *));                 //free
    extern BSTNODE *getBSTroot(BST *t);
    extern void    setBSTroot(BST *t,BSTNODE *replacement);
    extern void    setBSTsize(BST *t,int s);
    extern BSTNODE *insertBST(BST *t,void *value);
    extern BSTNODE *findBST(BST *t,void *value);
    extern BSTNODE *deleteBST(BST *t,void *value);
    extern BSTNODE *swapToLeafBST(BST *t,BSTNODE *node);
    extern void    pruneLeafBST(BST *t,BSTNODE *leaf);
    extern int     sizeBST(BST *t);
    extern void    statisticsBST(BST *t,FILE *fp);
    extern void    displayBST(BST *t,FILE *fp);
    extern void    displayBSTdebug(BST *t,FILE *fp);
    extern void    freeBST(BST *t);
    #endif
```

The BST and BSTNODE structures and methods should all be placed in *bst.c*.

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.

- *newBST* - The constructor is passed four functions, one that knows how to display the generic value stored in a node, one that can compare two generic values, one that knows how to swap the two generic values held by *BSTNODE*s (the *swapper* function is used by *swapToLeafBST*), and one that knows how to free a generic value. If the swapper function is NULL, then the constructor should store its own swapper function.
- *insertBST* - This method inserts a value into the search tree, returning the inserted BSTNODE.
- *setBSTroot* - This method updates the root pointer of a *BST* object. It should run in constant time.
- *deleteBST* - This method is implemented with a call to the swap-to-leaf method followed by a call to the prune-leaf method, returning the pruned node.

- *findBST* - This method returns the node that holds the searched-for value. If the value is not in the tree, the method should return null.
- *swapToLeafBST* - This method takes a node and recursively swaps its value with its successor's (preferred) or its predecessor's until a leaf node holds the original value. It calls the *BST*'s swapper function to actually accomplish the swap, sending the two nodes whose values need to be swapped.
- *pruneLeafBST* This method detaches the given node from the tree. It does not free the node nor decrement the count, however. Those tasks are left to the deletion method.
- *sizeBST* - This method returns the number of nodes currently in the tree. It should run in amortized constant time.
- *statisticsBST* - This method should display the number of nodes in the tree as well as the minimum and maximum heights of the tree. It should run in linear time. Example:

  ```
  Nodes: 8
  Minimum depth: 2
  Maximum depth: 4
  ```

  The minimum depth of a tree is the minimum number of steps from the root to a node with a null child. The maximum depth is similar. The depths of an empty tree are -1.
- *displayBST* - The display method performs a pre-order traversal of the tree. At any given node, the method displays the left and right subtrees, each enclosed with brackets, but only if they exist. A space appears in the output only to separare any existing subtrees (e.g. after the bracket for a left subtree) and the node value. An empty tree is displayed as `[]`. To display a node in the tree, the cached display function is passed the value stored at the node. No characters are to be printed before the first opening bracket or after the last closing bracket. This method should run in linear time.
- *displayBSTdebug* - This method displays a lever-order traversal of the tree, each level on its own line. Each line should have no preceeding whitespace and no trailing whitespace (other than a newline). A single space should separate entries on a line. No output should be generated for an empty tree. A line entry is generated by calling the cached display function. This method should run in linear time (HINT: use a queue).
- *freeBST* - This method walks through the tree, freeing the nodes that make up the tree. Then, the tree object itself is freed.
- *getBSTNODEvalue* - This method should return the value stored in the given search tree node.
- *getBSTNODEleft* - This method should return the left child of the given node, returning NULL if there is no left child.
- *setBSTNODEleft* - This method should set the left child of the given node. The former left child is not freed (it is assumed that the caller as gotten a handle on the former child be calling *getBSTNODEleft*.
- *freeBSTNODE* - If the freeing function is not NULL, then the method should free its generic value before freeing the node itself (with the system *free*).

The only local includes a *BST* module should have are *bst.h* and *queue.h* (needed by the BST display method).

Assertions

Include the following assertions in your methods:

- *newBST* - The memory allocated shall not be zero.
- *insertBST* - The memory allocated shall not be zero.

Testing your BST class

Modify the testing program found in the *singly-linked list class description* to work with a binary search tree.