A Simple Binary Search Tree Class

Version 1d

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>
#include "tnode.h"

typedef struct bst BST;

extern BST   *newBST(int (*c)(void *,void *));
extern void   setBSTdisplay(BST *t,void (*d)(void *,FILE *));
extern void   setBSTswapper(BST *t,void (*s)(TNODE *,TNODE *));
extern void   setBSTfree(BST *t,void (*)(void *));
extern TNODE *getBSTroot(BST *t);
extern void   setBSTroot(BST *t,TNODE *replacement);
extern void   setBSTsize(BST *t,int s);
extern TNODE *insertBST(BST *t,void *value);
extern void  *findBST(BST *t,void *key);
extern TNODE *locateBST(BST *t,void *key);
extern void  *deleteBST(BST *t,void *key);
extern TNODE *swapToLeafBST(BST *t,TNODE *node);
extern void   pruneLeafBST(BST *t,TNODE *leaf);
extern int    sizeBST(BST *t);
extern void   statisticsBST(BST *t,FILE *fp);
extern void   displayBST(BST *t,FILE *fp);
extern int    debugBST(BST *t,int level);
extern void   freeBST(BST *t);
#endif
```

The BST structure 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 a single function, one that knows how to compare two generic values. This function is known as a *comparator*.
- *setBSTswapper* The function passed to this method is used to swap the two generic values held by *TNODE*s. The *swapper* function is used by *swapToLeafBST* to a node's value with the value its predecessor/successor. Note, by default, the constructor should store its own swapper function. A call to *setBSTswapper* overrides the default method.
- *insertBST* - This method inserts a value into the search tree, returning the inserted *TNODE*.
- *setBSTroot* - This method updates the root pointer of a *BST* object. It should run in constant time.
- *deleteBST* - This method returns the value containing the searched-for key. The tree node that held the value is removed from the tree. This method also decrements the size. HINT: swapping the value to a leaf and pruning that leaf is an easy way to implement delete.
- *findBST* - This method returns the value with the searched-for key. If the key is not in the tree, the method should return null.
- *locateBST* - This method returns the tree node holding the searched-for key. If the key is not in the tree, the method should return null.
- *swapToLeafBST* - This method takes a node and recursively swaps its value with its predecessor's (preferred) or its successor'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 size, however.

- *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 prints a level-order traversal of the tree, each level on its own line. Each line should start with the level number and have no preceding whitespace and no trailing whitespace (other than a newline). A single space should separate entries on a line. A line entry is generated by calling the cached display function. The root value is tagged with an `X`, a left child with an `L`, a right child with an `R`, and a leaf with an equals sign. Unlike the other tags, the equal sign should precede the node value. All entries are tagged with their parents' value. No output should be generated for an empty tree. This method should run in linear time (HINT: use a queue). Here is a sample display:

  ```
  0: 20(20)X
  1: =7(20)L =33(20)R
  ```

  An empty tree should just print `0:` followed immediately by a newline.
- *debugBST* - If the debug level is one, two, or three, the display method should print an in-order, pre-order, or post-order traversal, respectively, 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 is printed by *displayBST* only to separate any existing subtrees (e.g. after the bracket for a left subtree) from 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. Here is a sample display:

  ```
  [[7] 20 [33]]
  ```
- *freeBST* - This method walks through the tree, freeing the nodes that make up the tree (and their values, if appropriate). Then, the tree object itself is freed.

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 *dynamic array class description* to work with a binary search tree.

Change log

`1d`	changed the second description of findBST to locateBST
`1c`	added the locate method
	added more debug levels
	added an example of displaying an empty tree
`1b`	flipped the logic of a display and a debugged display
`1a`	fixed prototypes for set display,free,swapper