Due : Wednesday, June 1st at 11:59 in p9 of cs40a.
New concepts: recursion, binary search tree, assert.
Files to be submitted: authors.csv, BinarySearchTree.cpp, BinarySearchTree.h, and Makefile.

You are to write a templated BinarySearchTree class that will implement the insert(), find(), remove(), inOrder(), and postOrder() functions called from the main() I have written. You will also need to write a BSTNode class that has BinarySearchTree as a friend. All of the functions called from main() are not recursive, but they all call private recursive versions with the same names that also take the root of the tree as a parameter.

A BSTNode is like a ListNode, but has two "next" pointers, named left and right. Like a linked list, a binary search tree (referred to as BST from now on) only has a single data member--a BSTNode* named root. A "binary tree" can be defined as a (possibly empty) collection of BSTNodes that consists of a distinguished node, called the root and zero, one, or two binary (sub)trees at which the root node points with its left and right variables. Since a "binary tree" is defined in terms binary trees, it is a recursive definition that lends itself to recursive functions. A BST is a binary tree with the additional property that all data in the nodes in the left subtree of a BSTNode have data that is less than or equal to the data in that node, and all data in the nodes in the right subtree of the BSTNode have data that is greater than or equal to that data. The STL map, multimap, set, and multiset store their information in BSTs.

To write this program, first write a simple Makefile. Next, write the BinarySearchTree class with the necessary function stubs to make main() compile. Then write the body of the functions in the order described below. By adding the asserts() as you go, you can check whether you program is working properly. Every recursive function tests for a NULL BSTNode* as one base case. Some functions have more than one base case to stop the recursion.

Inserting into a BST is straightforward. Start at the root, and go left if the value to be inserted is less than the data in the current a node, or right if it is greater. Continue until you reach a NULL pointer in the direction you wish to go, and assign that pointer to a new BSTNode that holds the value. This new node would have both its left and right pointers set to NULL. Nodes that have no children are called "leaves". We typically draw BSTs upside down, with the root at the top, and the leaves at the bottom.

An inorder traversal is prints the value(s) of the left subtree, then the value of the current node, and then the value(s) of the right subtree. Other than checking that BSTNode pointer is not NULL, this function has only three lines.

A postorder traversal is almost identical to an inorder traversal, except that the order is: left subtree, right subtree, and finally the node itself.

Finding a particular value in a BST follows the same pattern as an insert, except that if comes upon a NULL before finding the value, then it knows that the value is not in the BST.

Removing is much more complex because there are four possibilities to deal with:

1. The value isn't in the tree. Then do nothing to the tree.
2. The node that holds the value has no children, i.e. it is a leaf. Then just delete the node.
3. The node that holds the value has only one child. Then "promote" the child by having the parent of the node point at the child of the node.
4. The node that holds the value has two children. Then use a findMin() function to return the address of the BSTNode in the right subtree of the node that contains the smallest value in that subtree. Set the value of the current node to that minimum value, and then call the recursive remove() again, but this time pass the right pointer of the current node, and that minimum value as its parameters.

You can learn about binary search trees at https://en.wikipedia.org/wiki/Binary_search_tree. Please be forewarned that there is lots of code for the assignment available on the web, but if you don't write your own versions, MOSS will definitely catch you, and I will report you to the SJA. There will be a recursive problem on the final, and just relying on ideas from the web will assure you of losing many points on the final. Try writing them yourself. Other than remove() they are quite short.

You can find file*.txt, my BST.out, my BST2.out, and main.cpp in ~ssdavis/40/p9.

Further specifications:

1. Like the STL set and map classes, your BST may only rely on the "<" operator for comparisons.
2. The recursive insert function must be written as: insert(BSTNode<T> *t, const T &value).
3. The recursive remove function must be written as remove(BSTNode<T> * &t, const T &value). You will note that it is passing the BSTNode pointer by reference. This allows you to change the parent's value in the called function. So if I call remove (t->left, 8); then in the called function I can change what its parent's left pointer is pointing at! By copying the current value of t to a temporary variable, you can assign t the address of a child, and then delete the old t.

4. The BinarySearchTree destructor will need to call the recursive makeEmpty() function that deletes the children of a node after makeEmpty() has been called with them as parameters.
5. assert is available in <cassert> and is discussed on pp. 241-2 in the text.
   5.1. You must have assert() statements in four places in your program
      5.1.1. In the non-recursive remove(), to test if root is NULL.
      5.1.2. In the recursive remove(), to test if the value is in the tree.
      5.1.3. In the non-recursive find(), to test if root is NULL.
      5.1.4. In the recursive insert(), to test if the inserted value is a duplicate.
   5.2. Though the line numbers will not match mine, the rest of the error message produced by assert must match mine.
6. Makefile.
   6.1. Your Makefile should have an "all:" rule that is simply dependent on both BST.out and BST2.out, but with no action indicated on the second line.
   6.2. BST.out is created from a main.o that is compiled with NDEBUG defined using the D option of g++ so that assert() does not work.
   6.3. BST2.out is created from a main2.o that is compiled without NDEBUG defined so that the assert() will work. You will have the linking rule dependent on main2.o. main2.o will be created from main.cpp using both the g++ -o option, and the g++ -c option.

```
int main(int argc, char** argv)
{
  BinarySearchTree<int> tree;
  ifstream inf(argv[1]);
  char operation;
  int value;

  while(inf >> operation >> value)
  {
    switch(operation)
    {
      case 'I' : tree.insert(value); break;
      case 'R' : tree.remove(value); break;
      case 'F' : tree.find(value); break;
      case 'i' : tree.inOrder(); break;
      case 'p' : tree.postOrder(); break;
    }  // switch()

    inf.ignore(100, '\n');
  } // while more in file

  return 0;
} // main())

[ssdavis@lect1 p9]$ cat file1.txt
I18
I23
I7
i0
p0
[ssdavis@lect1 p9]$ BST.out file1.txt
7, 18, 23,
7, 23, 18,
[ssdavis@lect1 p9]$ BST2.out file1.txt
7, 18, 23,
7, 23, 18,
[ssdavis@lect1 p9]$
```

```
[ssdavis@lect1 p9]$ cat file2.txt
I8
I43
I73
I23
I67
R43
i0
p0
R25
i0
[ssdavis@lect1 p9]$ BST.out file2.txt
8, 23, 67, 73,
23, 73, 67, 8,
Not found.
8, 23, 67, 73,
[ssdavis@lect1 p9]$ BST2.out file2.txt
8, 23, 67, 73,
23, 73, 67, 8,
BST2.out: BinarySearchTree.cpp:172: void BinarySearchTree<T>::remove(BSTNode<T>*&, const
T&) [with T = int]: Assertion `t != __null' failed.
Abort (core dumped)
[ssdavis@lect1 p9]$ cat file3.txt
F7

[ssdavis@lect1 p9]$ BST.out file3.txt
Not found.
[ssdavis@lect1 p9]$ BST2.out file3.txt
BST2.out: BinarySearchTree.cpp:35: void BinarySearchTree<T>::find(const T&) const [with T
= int]: Assertion `root != __null' failed.
Abort (core dumped)
[ssdavis@lect1 p9]$ cat file4.txt
I8
I8
p0

[ssdavis@lect1 p9]$ BST.out file4.txt
Found duplicate.
8,
[ssdavis@lect1 p9]$ BST2.out file4.txt
BST2.out: BinarySearchTree.cpp:101: void BinarySearchTree<T>::insert(BSTNode<T>*, const
T&) [with T = int]: Assertion `value < t->data || t->data < value' failed.
Abort (core dumped)
[ssdavis@lect1 p9]$ cat file5.txt
R12
I13
I11
I9
i0
p0
[ssdavis@lect1 p9]$ BST.out file5.txt
Not found.
9, 11, 13,
9, 11, 13,
[ssdavis@lect1 p9]$ BST2.out file5.txt
BST2.out: BinarySearchTree.cpp:162: void BinarySearchTree<T>::remove(const T&) [with T =
int]: Assertion `root' failed.
Abort (core dumped)
[ssdavis@lect1 p9]$
```