CSCI 340 **Computer Assignment 7** **Fall 2017**

(10 points)

Binary Search Trees and Tree Sort

For this computer assignment, implement a derived class (as a template) to represent a binary search tree. Since a binary search tree is also a binary tree, implement your binary search tree class from the base class of binary trees, which you implemented in your previous assignment.

The definition of the derived class of a binary search tree (as a template) is given as follows:

```
template < class T >
class binSTree : public binTree < T > {
public:
    void insert ( const T& x );                  // inserts node with value x
    void remove ( const T& x );                  // removes node with value x
private:
    void insert ( Node < T >*&, const T& );      // private version of insert
    void remove ( Node < T >*&, const T& );      // private version of remove
    Node < T >* pred ( Node < T >* );            // returns predecessor of node
};
```

Where x is the target value for insert ( ) and remove ( ) functions. These *public* functions simply call their *private* versions.

The *private* versions of insert ( ) and remove ( ) functions can be implemented as *recursive* functions. When you implement the remove ( ) function, consider three possible cases: the node to be removed (1) is a leaf; (2) has only one child; and (3) has two children. In the first case, simply delete the node. In the second case, bypass the node making a new link from its parent to its child, and delete the node. In the third case, find the predecessor of the node – first move to the left child of the node, and then move to rightmost node in the tree, replace the value of the node with the value of its predecessor, delete the predecessor, and update the link to the deleted node, where the *private* non-recursive member function pred ( ) can be used to return the predecessor of a node.

To test your derived class, the source file of a driver program, prog7.cc, is supplied to you, which is in directory: ~cs340/progs/17f/p7. This program generates two sets of random integers in the range [0, R] with R = 1000, and inserts these numbers of the first set in a vector of integers and the numbers of the second set in another vector of integers, and inserts the numbers of the first set in a binary search tree. The program searches for each number in the second vector in the binary search tree before and after the numbers in the second vector is removed from the tree. At the end, it prints the following values for the binary search tree: the number of nodes and the height of the tree. Finally, it prints the data values in the binary search tree in ascending order before and after the removals, since traversing a binary search tree inorder results a sorted list of values.

To get the size of a binary search tree, add the *public* and *private* versions of the size ( ) function in your binTree class with the prototypes unsigned size ( ) const and unsigned size (Node < T >*) const, respectively.

Put the implementation of your binary search tree class in a separate header file, say binSTree.h. Modified version of the class Node from the previous assignment includes the binTree and binSTree classes as *friends*, so the objects of these two classes can access the *private* members of the class Node. Include the header file Node.h in your header file binTree.h, and include the header file binTree.h in your header file binSTree.h. Make it sure that each header file is included only once, so the contents of each header file should be guarded as follows:

```
// include system header files
using namespace std;

#ifndef a–const–value
#define a–const–value
// statements in the header file
#endif
```

For each header file, use a different const–value.

To compile and link the driver program with the system library routines, execute: Make N=7. To test your binary–search–tree class, execute: Make execute N=7. This command executes the driver program and displays the output as well as any error messages both on the terminal screen and in prog7.out. After you are done with the program, you don't need its object and executable files any more. To delete them, execute: Make clean.

The correct output of this program is in file prog7.out, which is in the same directory with prog7.cc. If the program fails to generate the correct output, then to debug the program, copy the file prog7.cc in your own directory, uncomment some of the statements in that file, recompile and link the modified program, and then execute again.

<u>Note</u>: In *public* versions of insert ( ) and remove ( ), you'll get compile errors when you pass the root as an argument to these functions. To eliminate compile errors, you need to pass the root as this–>root.

When your program is ready, submit its header files to your TA by executing:
mail_prog.340 binTree.h binSTree.h.