CSCI 340 **Computer Assignment 6** **Spring 2018**
(10 points)

Binary Search Tree Class

For this computer assignment, implement a derived class 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.

You are required to implement the binary search tree class in `assignment6.cc` file. You will need `assignment6.h` and `assignment6main.cc`, which are fully implemented. `assignment6.cc` already contains necessary headers. All files are available at `/home/turing/mhou/public/csci340spring2018.`

The definition of the derived binary search tree class is given here to facilitate the following description:

```
class BST : public binTree {
    public:
        BST() : binTree() {}          // constructor
        void insert( int );           // insert an item in the tree
        bool search( int );           // search an item in the tree
        bool remove( int );           // remove an item in the tree
                                      // returns true when successful
        int  sumLeftLeaves();         // return the sum of values
                                      // of left leaves

    private:
        void insert( Node*&, int );  // private version of insert(int)
        bool search( Node*&, int );  // private version of search(int)
        bool remove( Node*&, int );  // private version of remove(int)
        int  sumLeftLeaves( Node*& n); // private version of sumLeftLeaves
};
```

The above *public* functions simply call their *private* versions. The *private* versions of `insert (), remove (), search()` and `sumLeftLeaves ()` functions can be implemented as *recursive* functions. You can assume there are no identical numbers to be inserted into the tree.

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 immediate predecessor of the node – first move to the left child of the node, and then move to rightmost node in the sub-tree, replace the value of the node with its immediate predecessor, and delete the predecessor. The pseudo-code is shown below.

```
BinarySearchTree-Remove ( Node n, int x)
1 if n is empty
2     stop
3 if n's data is greater than x
4     recursive call to BinarySearchTree-Remove ( n's left link, x)
```

```
5 if n's data is less than x
6     recursive call to BinarySearchTree-Remove ( n's right link, x)
7  if n has two non-empty children
8     pred ←find n's immediate predecessor
9     copy pred's data to n
10    recursive call to BinarySearchTree-Remove( n's left link, pred's data)
11 else if n is leaf
12    delete n
13    n ← null
14 else // n has one child
15    temp ← n
16    n ← n's only child
17    delete temp
```

Programming Notes:

1. You need to copy your `assignment5.h` and `assignment5.cc` files into your current folder for `assignment 6`. You may need to add a line of code in `assignment5.h` to make `BST` a friend class of your `Node` class if necessary. You need to comment out the statement "`#define BINTREE_MAIN`" in `assignment5.cc` since there is a new `main` method in `assignment6main.cc` file.

2. You need to copy the three files `assignment6.h`, `assignment6.cc`, and `assignment6main.cc` from the directory shown in the last page. You need to add the implementation of `BST` class in `assignment6.cc`.

3. In the final version of your assignment, you are not supposed to change existing implementation, including the `main` method, provided to you in the original files `assignment6.h` and `assginment6main.cc`.

4. To compile the source file, execute "`g++ -Wall assignment5.cc assignment6.cc assignment6main.cc –o assignment6.exe`". To test your program, execute "`./assignment6.exe &> assignment6.out`". You can find the correct output of this program in file `assignment6.out` in the directory shown in the last page.

5. Add documentation to your source file.

6. Prepare your `Makefile` so that the TA only needs to invoke the command "`make`" to compile your source file and produce the executable file `assignment6.exe`. Make sure you use exactly the same file names specified here, i.e. `assignment6.h, assignment6.cc, assignment6main.cc` and `assignment6.exe`, in your `Makefile`. Otherwise your submission will get 0 points.

7. When your program is ready, submit files `assignment5.h, assignment6.h, assignment5.cc, assignment6.cc, assignment6main.cc` and `Makefile` to your TA by following the Assignment Submission Instructions.