CSCI 340 **Computer Assignment 5** **Spring 2018**
(10 points)

Binary Tree Class

For this computer assignment, you are to write a C++ program to implement classes to represent a binary tree (of integers). You are required to implement assignment5.h and assignment5.cc files. Both of them are partially implemented. assignment5.h already contains the class definition of `binTree`, and assignment5.cc already contains implementation of the `main()` function. Both files are available at `/home/turing/mhou/public/csci340spring2018.`

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

```
class binTree {
   public:
      binTree ( );                              // default constructor
      int height ( ) const;                     // returns height of tree
      unsigned size () const;                   // return size of tree
      virtual void insert ( int );              // inserts a node in tree
      void inorder( void(*)(int) );             // inorder traversal of tree
      void preorder( void(*)(int) );            // preorder traversal
      void postorder( void(*)(int) );           // postorder traversal
   protected:
      Node* root;                               // root of tree
   private:
      int height(Node*) const;                  // private version of height()
      unsigned size(Node*) const;               // private version of size()
      void insert (Node*&, int);                // private version of insert()
      void inorder( Node*, void(*)(int) );   // private version of inorder()
      void preorder( Node*, void(*)(int) ); // private version of preorder()
      void postorder( Node*, void(*)(int));// private version of postorder()
};
```

Most of the *public* member functions of the `binTree` class call *private* member functions of the class (with the same name). All of these *private* member functions can be implemented as either *recursive* or *non-recursive*, but clearly, *recursive* versions of these functions are preferable because of their short and simple implementations in code.

Because of information hiding, a client is not permitted to access the binary tree directly, so the *root* of the tree is kept *protected* (not *private* because of future implementations of *derived* classes from the *base* class of the `binTree`), so it cannot be passed as an argument to any of the *public* functions of the tree. It is essential to have *private* utility functions, which act as interface between a client and the tree.

The private `insert()`, `size()`, `height()`, and `inorder()` functions of the `binTree` class are described as following. `preorder()` and `postorder()` are similar to `inorder()`.

`insert ( Node*& r, int x )` : This function is used to insert a node with the data value `x` in a binary (sub-)tree at root `r`, applying the following technique: if the tree is empty, then the new node will be the *root* of the tree with the value `x;` otherwise, `x` is inserted in the left

or right sub-tree of `r`, depending on the sub-trees' **sizes**. If the size of the right sub-tree is less than the size of the left sub-tree, `x` is inserted in the right sub-tree; otherwise `x` is inserted in the left sub-tree.

`size(Node* r) const` : This function returns the number of nodes in the tree rooted at `r`. If the tree is empty, the size is 0.

`height(Node* r) const` : This function returns the height of the tree rooted at `r`. If the tree is empty, the size is -1.

`Inorder ( Node* r, void(* p)(int) )` : This function traverse the tree rooted at `r`. `p` is the "visit" operation on each node. To visit `r`, simply invoke `p(r->data)`.

Programming Notes:

1. You need to add the definition of the `Node` class in the file assignment5.h. You need to make `binTree` class a friend of the `Node` class.

2. The file `assignment5.cc` also contains some constants, global variables, and the implementation of function `display()`, which is invoked by `main()`. You need to add the implementation of `binTree` class (and `Node` class if necessary) in `assignment5.cc`.

3. Include any necessary headers and add necessary global constants.

4. You are not allowed to use any I/O functions from the C library, such as `scanf` or `printf`. Instead, use the I/O functions from the C++ library, such as `cin` or `cout`.

5. In the final version of your assignment, you are not supposed to change existing code, including the `main` method, provided to you in the original files `assignment5.h` and `assignment5.cc`.

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

7. Add documentation to your source file.

8. Prepare your `Makefile` so that the TA only needs to invoke the command "`make`" to compile your source file and produce the executable file `assignment5.exe`. Make sure you use exactly the same file names specified here, i.e. `assignment5.h, assignment5.cc` and `assignment5.exe`, in your `Makefile`. Otherwise your submission will get 0 points.

9. When your program is ready, submit your header file `assignment5.h`, source file `assignment5.cc` and `Makefile` to your TA by following the Assignment Submission Instructions.