Lab 4: BST

By Dr. Yingwu Zhu, Seattle University

1. <u>Goals</u>

This lab aims to familiarize you with BST and its operations. You need to be able to implement a BST that efficiently support its operations as a dynamic set discussed in class.

2. <u>Design</u>

In this lab, you need to implement the following basic operations:

- Constructor
- Destructor
- *bool empty()* which return true if the BST is empty; false otherwise
- *void recurs_preorder(), void recurs_inorder()* and *void recurs_postorder()* which recursively traverse the tree in preorder, inorder and postorder, respectively.
- *bool search(ElementType x)* which returns true if *x* exists on the list; false otherwise
- *void nonrecurs_preorder()* that traverses the tree in preorder with iterative algorithm.
- *void insert(ElementType x)* which inserts a new element x into the list and maintains the non-decreasing order

Note: **typedef int ElementType;**

3. <u>Implementation</u>

Now, it is time for you to come up with the class declaration for **BST** in **bst.h**

Under your **lab4** directory, please complete the header file **bst.h**

Stop here! Complete the header file! (10 points)
==

```
/*
   Author: xxxxx (your name)
   Program: bst.h
   Last Modification: xxx (date)
*/

/*
   Description: this header file is for BST
*/
#ifndef   _MY_BST_H
#define   _MY_BST_H
#include <iostream>
using namespace std;
typedef    int    ElementType;
class BST {
   private:
            class Node {
                  public:
                      ElementType  data;
                      Node* left;
                      Node* right;
                  public:
                       Node() : left(NULL), right(NULL) {  }
                       Node(ElementType x) : data(x), left(NULL), right(NULL) {  }
            };

            Node* root;
   public:
            BST();   //constructor
            ~BST();   //destructor
            bool  empty();
            bool  recurs_search(ElementType x);
            bool nonrecurs_search(ElementType x);
            void  recurs_preorder();
            void  recurs_inorder();
            void  recurs_postorder();
            void nonrecurs_preorder();
            void  insert(ElementType x);
};


#endif
```

Note: you may need to adapt this class declaration when necessary!

Next, you are going to implement the **BST** in **bst.cpp** step by step!

Step 1: Implement constructor, destructor & empty() (10 points)

Thinking: What should be done in constructor and destructor? When will they be called? How do you implement the destructor?

Step 2: Test the constructor, destructor & empty() (10 points)

In this step, you need to create a client program **test.cpp** and a **Makefile**. In **test.cpp**, you should test **BST**'s constructor and destructor.
Note: You should comment out those function that you have NOT implemented, in order to compile your program successfully. Also reuse your Makefile in the previous labs by modifying the involved file names.

Step 3: Implement insert() and simply test it (10 points)
Questions: How do you perform insertion? Will the BST property help on this insertion operation?

Step 4: Implement three recursive traversals and test if they work correctly, combined with insert() (10 points)

Step 5: Implement non-recursive preorder traversal (10 points)
Hint: use STL stack class template, the off-the-shelf data structure.
Questions: How would you perform preorder traversal using the STL stack?

Step 6: Implement the search operation and test it (10 points)
Questions: can you implement it in both iterative and recursive versions?

==

4. <u>Extra lab exercises</u>

See the corresponding homework assignment for exercises.