COMP 2611 – Data Structures

Lab #5

A binary search tree (BST) is a binary tree where the keys stored at each node satisfy the *binary-search-tree property*:

- Let *x* be a node in a BST.
- If *y* is a node in the left subtree of *x*, then y->data < x->data.
- If *y* is a node in the right subtree of *x*, then y->data > x->data.

Download the `Lab5-Files.zip` file from myeLearning. The zip file contains a Dev-C++ project, `BinarySearchTree.dev`.

1. In `NodeTypes.h`, modify the declaration of the *BTNode* struct so that it stores the address of the parent of a node. In `BinaryTree.cpp`, modify the *createBTNode* function accordingly.

2. In `BinarySearchTree.cpp`, write a function *insertBST* that inserts a node in a binary search tree and returns the root of the binary search tree. The parent field of the new node must be set accordingly. The prototype of the *insertBST* function is:

```
BTNode * insertBST (BTNode * root, int data);
```

The algorithm to insert a node in a binary search tree is as follows (based on Slides 4-5 from Lecture 9):

```
newNode = createNode (data);

if (tree is empty)
      return newNode

curr = root

while (true) {
      if (newNode->data < curr->data) {
            if (curr has no left child) {
                  insert newNode as left child of curr
                  connect newNode to parent node
                  return root
            }
            curr = curr->left
      }
      else {
            if (curr has no right child) {
                  insert newNode as right child of curr
                  connect newNode to parent node
                  return root
            }
            curr = curr->right
      }
}
```

3. In `BinarySearchTree.cpp`, write a **recursive** function *containsBST* which determines if a key is present in a binary search tree. If the key is present, it returns the address of the node containing the key; otherwise, it returns NULL. The prototype of the *containsBST* function is:

```
BTNode * containsBST (BTNode * root, int key);
```

The four cases to consider are as follows (Slides 6-9 from Lecture 9):

(i) The binary search tree is empty.
(ii) The root of the binary search tree contains the key.
(iii) `key < root->data`.
(iv) `key > root->data`.

4. The *depth* of a node is the number of branches that must be traversed on the path to the node from the root. In `BinaryTree.cpp`, write the code for the function *nodeDepth* with the following prototype which finds the depth of the node passed as a parameter (Slide 13 from Lecture 9):

```
int nodeDepth (BTNode * node);
```

NB: If the address of the node passed as a parameter is NULL, return -1.

5. In `BinarySearchTree.cpp`, write a function with the following prototype to delete a **leaf** node from a binary search tree:

```
bool deleteLeafNode (BTNode * node);
```

The function must return *true* if the node was successfully deleted and *false*, otherwise (e.g., if it is a non-terminal node).

6. In the *main* function of `UsingBinarySearchTree.cpp`, write code to create the binary search tree in Figure 1:

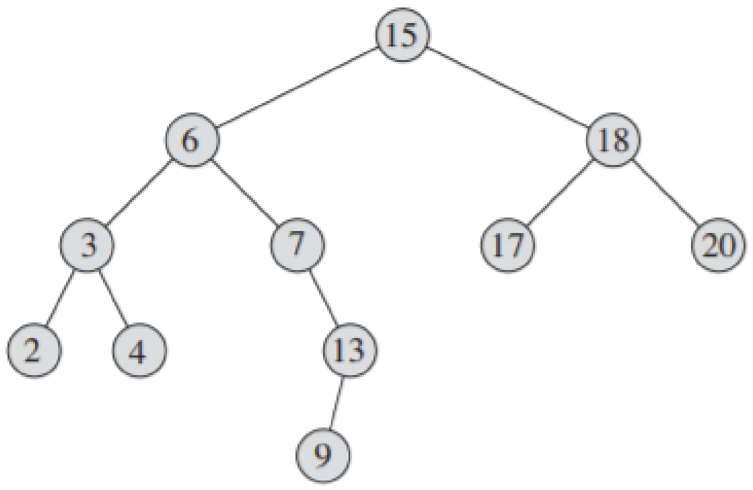

Figure 1: Binary Search Tree

7. Write code in the *main* function to perform a preorder, an inorder, and a postorder traversal of the binary search tree in Figure 1 using the traversal functions that were previously written in `BinaryTree.cpp`. Verify that the correct results are obtained.

8. Write code in the *main* function to test the *contains*, *nodeDepth*, and *deleteLeafNode* functions using the binary search tree in Figure 1. The *contains* function should be tested with keys 12, 13, and 17. The *nodeDepth* function should be tested with Node 15 and Node 13. The *deleteLeafNode* function should be tested with Node 13 and Node 17. Do an inorder traversal to verify that a leaf node was successfully deleted.

9. In `BinarySearchTree.cpp`, write a **recursive** function *insertBSTRec* which inserts a node in a binary search tree and returns the root of the binary search tree. The parent field of the new node must be set accordingly. The prototype of the *insertBSTRec* function is:

```
BTNode * insertBSTRec (BTNode * root, int key);
```

The three cases to consider are as follows:

(i) The binary search tree is empty: create a new node and return its address.
(ii) `key < root->data`: insert the key recursively in the left child of *root* and connect *root->left* to the address returned by the recursive call.
(iii) `key > root->data`: insert the key recursively in the right child of *root* and connect *root->right* to the address returned by the recursive call

In Case (ii) and Case (iii), the address returned by the recursive call (i.e., the subtree containing the new node) must be connected to its parent, *root*.

10. In the *main* function, create the binary search tree in Question 6 by calling the recursive function, *insertBSTRec*. Verify that the keys have been inserted correctly.

End of Lab #5