COMP 2611 – Data Structures

Lab #4

Binary Trees

Instructions

Download and unzip `Lab4-Files.zip`. Open the `BinaryTree.dev` project. The file, `BinaryTree.h`, contains the declaration of *BTNode*, a node in a binary tree:

```
struct BTNode {
   int data;
   BTNode * left;
   BTNode * right;
};
```

The file, `BinaryTree.cpp`, contains the code for the *createBTNode*, *preOrder*, *inOrder*, and *postOrder* functions from Lab #3. In this lab, you have to write some additional functions in `BinaryTree.cpp`.

NB: The `BinaryTree.dev` project now includes code for a Stack (`Stack.h` and `Stack.cpp`). The Stack is used in Question 3.

1. (a) The *moment* of a tree is the number of nodes in the tree. In `BinaryTree.cpp`, write a recursive function, *moment*, with the following prototype which returns the moment of the binary tree passed as a parameter:

```
int moment (BTNode * root);
```

(b) In `BinaryTree.cpp`, write a recursive function, *numOneChild*, with the following prototype, which returns the number of nodes in the binary tree passed as a parameter that have exactly one child:

```
int numOneChild (BTNode * root);
```

(c) In `BinaryTree.cpp`, write a recursive function, *numTerminal*, with the following prototype, which returns the number of leaf nodes in the binary tree passed as a parameter:

```
int numTerminal (BTNode * root);
```

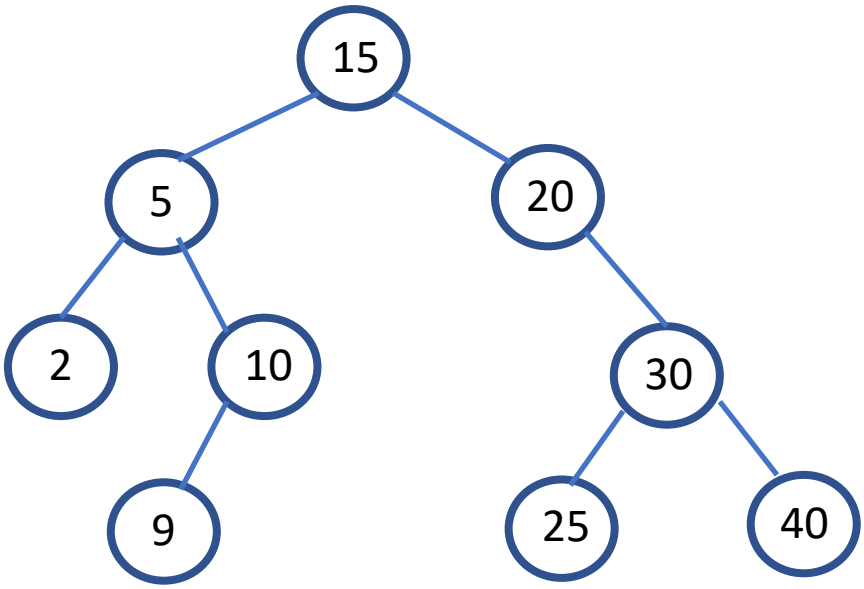
(d) In `BinaryTree.cpp`, write a function, *maximum*, with the following prototype, which returns the largest value in the binary tree passed as a parameter:

```
int maximum (BTNode * root);
```

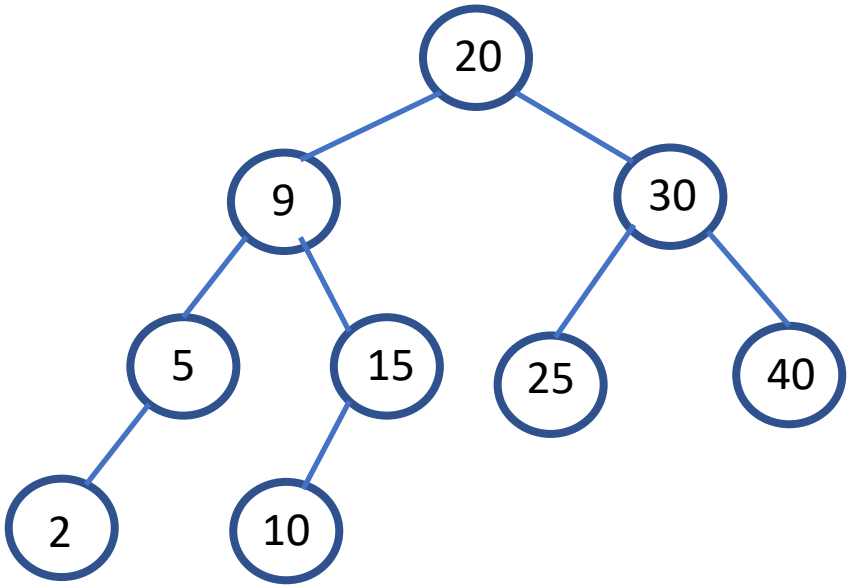
If the tree is empty, the function should return INT_MIN.

(e) In Using`BinaryTree.cpp`, write code to test the *moment*, *numOneChild*, *numTerminal*, and *maximum* functions using the binary tree created in Lab #3. Verify that the correct answers are obtained.

2. (a) Give the inorder traversal of the following binary tree:



(b) Give the inorder traversal of the following binary tree:



(c) (i) State two observations about the inorder traversals of the binary trees in (a) and (b).

(ii) Study the binary trees in (a) and (b) carefully. Where is the smallest value and the largest value located in each tree?

3. Previously, the preorder, inorder, and postorder traversals of a binary tree were obtained using recursion. We will now write a non-recursive version of the inorder traversal.

Recall that an inorder traversal goes left → root → right. When we go down the left subtree of the root, we will need some way to come back up to the root. This can be repeated at many levels, and we must come back up the same way that we went down to the subtrees.

A stack can be used to store the nodes that were met on the way down, but not visited. On the way back up, the nodes are popped from the stack in the reverse order that they were met. The following is an algorithm for a non-recursive inorder traversal using a stack (Kalicharan, 2008):

```
initialize stack S to empty
set curr = root
set finished = false
while (not finished) {
       while (curr != null) {
              push curr onto S
              set curr = left (curr)
       }
       if (S is empty)
              set finished = true
       else {
              pop S into curr
              visit curr
              set curr = right (curr)
       }
}
```

(a) Write the code for the non-recursive inorder traversal in `BinaryTree.cpp`.

(b) Test the non-recursive inorder traversal using the binary tree that was created in Lab #3 (In `UsingBinaryTree.cpp`).

(c) What modification/s to the algorithm is/are necessary to obtain a non-recursive preorder traversal?

(d) Write and test the code that performs a non-recursive preorder traversal of a binary tree.

END OF LAB #4