Data Structures & Algorithms

Lab Exercise 6 – Binary Tree Project (Part 2)

Due: to be demonstrated by week 19 (wk beginning 4-Feb)

Learning outcomes
At the end of this lab you should be able to:

- Implement different methods of traversing a binary tree.
- Describe and implement the height algorithm for a binary tree.

Q1. Write a global function (in same source file as main):

```
template <typename T>
void binaryPreorder(BinaryTreeNode<T> * node);
```

to perform an preorder traversal on a binary tree. For the 'visit' action, simply output the value of each node. Note that you need to check the node is not null before using it (for example, this will happen if an internal node has one left child node only, and we attempt to process the right child).

So, before you make a recursive call, for example to recurse down the left side of the tree, perform the check:

```
if (node->leftChild() != nullptr)
```

and similarly for recursing down the right side of the tree.

Call the function from inside main as follows:

```
binaryPreorder(myTree.root().node());
```

Q2. Now write a non-recursive version of inorder by initially creating an instance of an STL stack (a LIFO structure) and then following the algorithm below:

```
// n is a parameter to this algorithm (usually the root node)
Algorithm binaryInorder(n)

create an empty stack
while the stack is not empty OR n is not NULL {
    while n is not null {
        push n onto the stack
        let n = the left child of n
    }
    if n is null {
        let n = the node on top of the stack
        pop the stack
        visit n
        let n = the right child of n
    }
}
```

Short stack example:

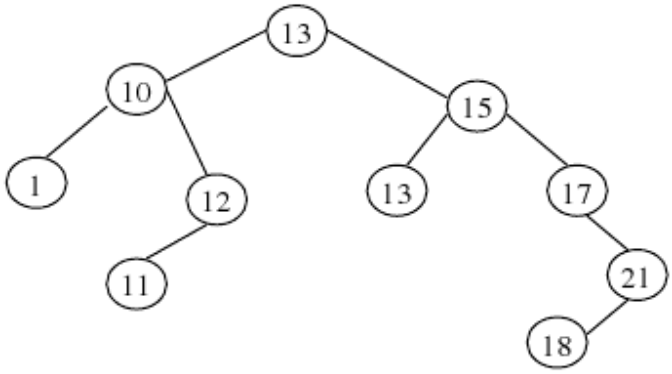
```
#include <stack>

int main() {
  stack<int> myStack;
  myStack.push(8);
  myStack.push(7);
  myStack.push(4);

  std::cout << myStack.size() << std::endl; // prints 3

  int x = myStack.top(); // returns 4
  myStack.pop(); // removes 4
}
```

Q3. Write a function to implement the `height()` algorithm for a binary tree. The height of a binary tree is computed as the node with the greatest depth (i.e. the deepest external node). For a tree with just one node, the root node, the height is defined to be 0. For example, the height of the tree below is 4.




```
// n is normally the root node
Algorithm height(n)

if (isInternal(n)){
         let x = height(leftchild(n))
         let y = height(rightchild(n))
         return greater(x,y) + 1
}
return 0
```

where:

`greater(X,Y)` – returns the greater of x and y.

At each node, the recursive algorithm will accumulate the height of the left and right sub trees and pass it to the parent by adding 1 to the greater one. However, if the node is external it will simply return 0 to parent.

1) Start by adding a pure virtual function to the SimpleTree interface.
2) Now write the implementation in the LinkedBinaryTree class.
3) Open the source file with the main() function. Create a binary tree with the above nodes and test your implementation of height.