Programming Exam Choice 18

Start by downloading the question 18 files from the moodle into a new directory on your machine. You will be changing the code in the question18.cpp file. You have a Makefile to handle building this code.

Part 1. 5 points

You will complete this function to take in a binary search tree, traverse it *in-order* and print its output to a doubly linked list. You MAY NOT rely on your doubly linked list's add_node code for this, though, because that sorts the data. Instead, you will have to build and connect the list nodes one by one, as you go.

```
void tree_to_list_inorder(const binary_tree_node* root_ptr,
        node*& head_ptr, node*& tail_ptr)
{

        // make one recursive call here

        // add a node to the list here, with the data in the root_ptr
        //      case 1: list was empty; head_ptr and tail_ptr must be updated here
        //      case 2: list is not empty, and we are adding a node after the current tail,
        //                  and updating the tail pointer

        // make another recursive call here

}
```

You should be able to verify that the numbers are printing in order. If you've done this as asked (without using add_node), then part 2 is a breeze.

Part 2. 5 points

With one small rearrangement, you should now be able to make lists for pre-order traversals in the next function:

```
void tree_to_list_preorder(const binary_tree_node* root_ptr,
        node*& head_ptr, node*& tail_ptr)
{


}
```

With a second rearrangement of your inorder code, you now can make lists for post-order traversals in the third function:

```
void tree_to_list_postorder(const binary_tree_node* root_ptr,
        node*& head_ptr, node*& tail_ptr)
{
```

```
}
```

Test your code on my trees and when you are satisfied, please upload your question18.cpp file to the moodle. Your TA may ask you to zip other files in as well.

Logic of problem laid out in comments:	50%
Code compiles with no errors or warnings:	10%
Code has no run time errors:	10%
Code gives correct answers for all inputs:	20%
Code is clean and easy to read:	10%