1 Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Homework09, put each programming exercise into its own subdirectory of this directory, zip the entire Homework09 directory up into the file Homework09.zip, and then submit this zip file to Homework #9.

Make sure that you:

- Follow the coding and documentation standards for the course as published in the MyClasses page for the class.
- Check the contents of the zip file before uploading it. Make sure all the files are included.
- Make sure that the file was submitted correctly to MyCLasses.

All non-templated class structures are to have their own guarded specification file (.h) and implementation file (.cpp) that has the same name as the class. All templated class structures are to be guarded and written entirely in their (.h) file. No inline coding in the class specification. In addition you must create a make file that compiles and links the project on a Linux computer with a Debian or Debian branch flavor.

2 Exercise #1

The tree sort is a simple sort once you have created a binary search tree. Simply take the elements of the array, load them into a binary search tree, then traverse the tree in an in-order fashion loading the elements of the tree back into the array. Since the reloading of the array will be done during an in-order traversal of the tree the array will be sorted.

1. Take the binary tree class from the example code and the sorts timing code you used to time the merge and quick sorts in a previous assignment.
2. Add two functions to the BinaryTree class,

   ```
   void loadInOrderRec(TreeNode *nodePtr, T A[], int &pos)
   ```

   ```
   void loadInOrder(T A[])
   ```

   The `loadInOrderRec` is a recursive function that does an in-order traversal of the tree and upon a node visit loads the element into the next position in the array. The function `loadInOrder` simply calls the recursive one on the root of the tree.
3. Add a templated function to the sorts that will take an array, load it into a binary search tree and then use the tree `loadInOrder` function to reload the array.

```
void treeSort(T A[], int size)
```

4. Test the tree sort on small arrays to make sure it is working correctly.
5. Use the timer code to compare the tree sort, quick sort, and the merge sort. That is, do the same analysis on the tree sort that you did in a previous assignment. Remember that for a good comparison all timings need to be done on the same machine.
   (a) Run the program on array sizes, 1,000,000, 2,000,000, 5,000,000, 10,000,000, 20,000,000, 30,000,000, ... elements. Keep increasing the size of the array by 10,000,000 elements each time until the timings go over 15 seconds.
   (b) Make a chart from the data that has the array size as the first column, the quick sort times as the second column, the merge sort times as the third column and the tree sort times in the fourth column. Let the x values be in the millions, so for the list above they would be 1, 2, 5, 10, 20, 30, ...
   (c) Create a line graph of sort times verses array size, have the array size on the horizontal axis and the sort times on the vertical axis. So the graph should have three lines all on the same chart. As before an XY scatter plot with line would probably work best.
   (d) From the time analysis, which algorithm is consistently better? Do the lines cross? This would indicate that one algorithm is better for smaller arrays but another is better for larger arrays. From the timing data, does there seem to be any consistencies between the algorithms? As an example, is one consistently 10 times slower than another one, are they all fairly close to each other on every run?
   (e) Put all the charts, graphs, and analysis into a document, export the document to PDF and include it with your code solutions to this exercise in the zip file.

3 Exercise #2

Add the following functions to the templated binary search tree and create a main program that tests all of them.

1. First do the fix we did in class to the `deleteNode` function so that if the node value is not found the tree would be unaltered and the program will not crash with a segfault.
2. Node Counter: Write a public member function `numberOfNodes()` that counts and returns the number of nodes in the tree. You will want to follow the design of having this non-recursive function call a recursive version that does the majority of the work, the recursive version is to be private.
3. Leaf Counter: Write a public member function `numberOfLeafNodes()` that counts and returns the number of leaf nodes in the tree. You will want to follow the design of having this non-recursive function call a recursive version that does the majority of the work, the recursive version is to be private.

4. Tree Height: Write a public member function `height()` that returns the height of the tree. The height of the tree is the number of levels it contains. You will want to follow the design of having this non-recursive function call a recursive version that does the majority of the work, the recursive version is to be private.

5. Clear: Write a public member function that destroys the entire tree and deletes all allocated memory, that is, no memory leaks. As with the above functions, if you call a recursive function to support this operation have the recursive function be private or protected.

6. Tree Assignment Operator and Copy Constructor: Write an overloaded assignment operator and a copy constructor. As with the above functions, if you call recursive functions to support these operations have the recursive functions be private.

The following program gives the output below.

```cpp
#include <cstdlib>
#include <ctime>
#include <iostream>

#include "BinaryTree.h"

using namespace std;

template <class T> void TreeInfo(BinaryTree<T> &tree) {
    cout << endl;
    tree.PrintTree(5);
    cout << endl;
    cout << "Number of nodes in the tree = " << tree.numberOfNodes() << endl;
    cout << "Number of leaf nodes in the tree = " << tree.numberOfLeafNodes()
         << endl;
    cout << "Height of the tree = " << tree.height() << endl;
}

template <class T> void TreeInfoBV(BinaryTree<T> tree) {
    cout << "Tree Copy" << endl;
    TreeInfo(tree);
}

void div() { cout << "\n--------------------------\n\n"; }

int main() {
    srand(time(NULL));
    int size = 0;
    BinaryTree<int> tree;

    cout << "Empty Tree";
    TreeInfo(tree);
    cout << endl;

    cout << "Input number of elements to insert: ";
    cin >> size;

    for (int i = 0; i < size; i++)
        tree.insertNode(rand() % 100 + 1);

    TreeInfo(tree);
    div();

    BinaryTree<int> tree2 = tree;
    TreeInfo(tree2);

    div();

    for (int i = 0; i < 10; i++)
        tree.insertNode(rand() % 100 + 1);

    TreeInfo(tree);
    TreeInfo(tree2);

    div();

    BinaryTree<int> tree3;
    tree3 = tree2;
    TreeInfo(tree3);

    div();

    TreeInfoBV(tree2);

    div();

    for (int i = 0; i < 5; i++)
        tree2.insertNode(rand() % 100 + 1);

    tree = tree3 = tree2;
    TreeInfo(tree);
    TreeInfo(tree2);
    TreeInfo(tree3);

    return 0;
}
```

```
Empty Tree

Number of nodes in the tree = 0
Number of leaf nodes in the tree = 0
Height of the tree = 0

Input number of elements to insert: 7

          52
     47
               46
          42
               22
19
     6

Number of nodes in the tree = 7
Number of leaf nodes in the tree = 4
Height of the tree = 4

--------------------------


          52
     47
               46
          42
               22
19
     6

Number of nodes in the tree = 7
Number of leaf nodes in the tree = 4
Height of the tree = 4

--------------------------


               89
                    87
                              68
                                   59
                         58
          52
     47
               46
          42
                         32
                    29
               22
19
               16
          7
     6
          1

Number of nodes in the tree = 17
Number of leaf nodes in the tree = 5
Height of the tree = 8

          52
     47
               46
          42
               22
19
     6

Number of nodes in the tree = 7
Number of leaf nodes in the tree = 4
Height of the tree = 4

--------------------------


          52
     47
               46
          42
               22
19
     6

Number of nodes in the tree = 7
Number of leaf nodes in the tree = 4
Height of the tree = 4

--------------------------
```

```
Tree Copy

          52
     47
               46
          42
               22
19
     6

Number of nodes in the tree = 7
Number of leaf nodes in the tree = 4
Height of the tree = 4

--------------------------


               64
          52
     47
               46
          42
                         35
                    28
               22
19
          12
     6
          5

Number of nodes in the tree = 12
Number of leaf nodes in the tree = 5
Height of the tree = 6

               64
          52
     47
               46
          42
                         35
                    28
               22
19
          12
     6
          5

Number of nodes in the tree = 12
Number of leaf nodes in the tree = 5
Height of the tree = 6

               64
          52
     47
               46
          42
                         35
                    28
               22
19
          12
     6
          5
```

```
Number of nodes in the tree = 12
Number of leaf nodes in the tree = 5
Height of the tree = 6
```