CS 280 | Spring 2017
Programming Assignment #4

Files (submit folder) due

- Monday, June 5, 2017
- 01:00 am

This assignment gives you some experience implementing a simple API for binary trees. You will also have another chance to become comfortable with recursive algorithms while manipulating binary trees. Because trees are a form of sequence container, the interface will be simpler than the interfaces we've seen for lists. In other words, since the container must remain sorted, there are no methods for adding/removing to/from the end/beginning or inserting at any arbitrary location. The task is to implement two classes named BSTree and AVLTree which clients can use to store data. The public interface for the BSTree class is straight-forward. The partial class definitions are below:

Sample command lines (for the templated version):

Microsoft:

```
cl -Fems.exe Driver.cpp -EHa -Za -W4 -WX -MTd -Zc:forScope -D_CRT_SECURE_NO_DEPRECATE
```

GNU:

```
g++ -o gnu.exe Driver.cpp -O -Wextra -Wall -ansi -pedantic -Werror -Wconversion
```

Borland:

```
bcc32 -w -w-8026 -w-8091 -ebor.exe Driver.cpp
```

The public interface will not be able to do the recursion. You'll need to do things like this with a private method:

```
template <typename T>
void BSTree<T>::insert(const T& value) throw(BSTException)
{
    // Call a private recursive method
  insert_item(root_, value);
}
```

Using the stack-based method demonstrated in class, you'd have something like this:

```
template <typename T>
void AVLTree<T>::insert(const T& value) throw(BSTException)
{
    // Create a stack to use with the private recursive method
  std::stack<typename BSTree<T>::BinTree *> nodes;
  insert_item(this->get_root(), value, nodes);
}
```

Notice the **`typename`** keyword and the *non-redundant* (read: required) use of **`this`** above. You could do this as well:

```
  insert_item(BSTree<T>::get_root(), value, nodes);
```

You'll need to implement this function as appropriate:

```
template <typename T>
bool AVLTree<T>::ImplementedBalanceFactor(void) const
{
    // Change to true if you used the efficient method and want extra credit
 return false;
}
```

Notes

1. The only public method that throws an exception is the *insert* method. There are two kinds of exceptions that can be thrown: out of memory and inserting a duplicate element. The exception class is provided.

2. To the user, there is no concept of an index or position in tree structures. The position of the element in the tree is determined by its value in relation to the other elements in the tree.

3. Allocate and free nodes using the C++ operators **new** and **delete**. (You don't need to use a memory manager, although in a real world efficient implementation, you would use one. We've already done this in another assignment and I don't think you need to do it again.).

4. The *find* method returns a boolean, indicating whether or not the item was found. There is a second parameter which will contain the number of comparisons performed to determine the outcome of *find*. **Make sure your counts match the counts from the sample driver to receive credit.**

5. Like all of our templated classes, you will include the implementation files in the header files.

6. The public *root* method simply returns the root of the tree. This allows the user to walk the tree. (Normally, we wouldn't want that, but for learning purposes, we would like to be able to examine the tree outside of the class, such as within the text or GUI driver.)

7. This first part of this assignment (implementing the *BSTree* class) is rather trivial since I have already shown you all of the code and explained it. That code is available to you from the web site. You just need to put it in a class and then templatize it.

8. You'll notice that AVL.h includes <stack>. For this assignment you can use std::stack from the STL instead of having to write your own. You'll need a stack to implement the simple balancing algorithm as it was demonstrated in class. If you'd like, you can balance the tree recursively, which doesn't require a separate std::stack.

9. You can't change the public interface at all. That means you can't add, remove, or change any *public* method.

10. There is no extra credit on this assignment so return false from the ImplementedBalanceFactor

Implementation Steps

If you break down the assignment into manageable pieces, it isn't that difficult. However, if you decide to start immediately with the sample templated driver, you will get nowhere fast. You should solve the problem in pieces. This is a smarter approach to implement this assignment (or any assignment, for that matter).

1. Implement a non-templated (using integers) BSTree class and run some tests on it to make sure it works. This should only take a short amount of time since I've provided all of the code for this on the web pages.

2. Derive a non-templated AVLTree from BSTree and run some tests on it.

    a. Implement the balancing algorithm from the pseudocode I demonstrated in class. **Copy and paste the pseudocode into your code and use that to guide your implementation.**

3. Convert the classes into templatized classes and run your same tests on it. (You'll need to modify your driver)

4. Try your completed code with the sample templated driver that I provided.

Testing

As always, testing represents the largest portion of work and insufficient testing is a big reason why a program receives a poor grade. (My driver programs take longer to create than the implementation file itself.) Sample drivers program for this assignment are available. You should use the driver program as an example and create additional code to thoroughly test all functionality with a variety of cases. (Don't forget stress testing.)

What to submit

You must submit your program files (header and implementation files) and the project
and solution file if you make in a .zip file to the appropriate Moodle page by the due date and time.

BSTree.h
AVLTree.h

The header files. No implementation code is allowed. The public interface must be
exactly as described above.

BSTree.cpp
AVLTree.cpp
The implementation files. All implementation code goes here.