**Programming Assignment**

**Binary Tree**

**FAQ**

1. **Question: You said that we can't call new in our tree classes. How do we construct our own ObjectAllocator if we can't call new.**

**Answer:** I meant that you can't use **new** to allocate any memory for nodes (or anything else). The only place that you need to call **new** is:

* + when you are constructing your own allocator. (You aren't allocating any nodes.)
  + when you call placement **new** to construct the node in the memory returned from the allocator.

All nodes that you create must be created via the ObjectAllocator. If you call **new** for anything but the items above, your will lose points. Realize that this isn't a limitation for your program because you don't need to use **new** except where specified above. I would expect to see only 2 or 3 occurrences of the keyword **new** in your entire program. (You **DID** create a single function to allocate a node instead of calling the allocator from several places in your code, didn't you?)

1. **Question: I'm getting weird errors from the compilers when I try to build my code. The errors say something about a bad token '<'. What's wrong?**

**Answer:** You probably are including the implementation file twice. Remember, it's a templated class, so you have to include the implementation with the header file. Since it's already included in the header, adding it to the project or on the command line causes the code to be added twice. See the command line posted for an example of correct usage.

1. **Question: I'm making a function that returns a BinTreeNode\* but the compiler is complaining that it doesn't know what a BinTreeNode is. What's wrong?**

**Answer:** You probably forgot to use the keyword **typename** with the return value. Here's an example of how you would specify a function called make\_node that creates a node:

**template** <**typename** T>

**typename** BStree<T>::BinTreeNode\* BStree<T>::make\_node(**const** T& Item) **const** **throw**(BSTException)

The keyword in **RED** above is probably what's missing. Since BSTree is a templated class, the compiler doesn't know the type of T yet, so it can't look for BinTreeNode. Since it can't look it up, it *assumes* that BinTreeNode is a member variable of BSTree, which doesn't make sense when the compiler expects to find a type. (It is the return-**type** afterall). In order to tell the compiler that BinTreeNode is a type and not a member variable, use the **typename** keyword.

Refer to your CS225 notes for more information.

1. **Question: My ObjectAllocator isn't working correctly. How do I implement this assignment without a working allocator?**

**Answer:** Use the [dummy allocator](https://distance.sg.digipen.edu/file.php/1659/Assignment4/dummy_oa.zip) that I provided. It's just a wrapper around new/delete, but it contains everything you need to implement this assignment properly. Your stress tests won't be as fast, but don't worry about that. I will be using the dummy allocator when I grade your code.

1. **Question: I'm not getting the same number of compares that your output shows. For example, in an empty tree, I have 0 compares, but the output shows 1. Why is that?**

**Answer:** Because compares is not actually counting comparisons. It's counting the number of times the recursive method was called. So, when you first check the tree and find it empty (NULL), you haven't actually compared the value in the node with the value you are looking for. However, you did have to check (compare with NULL) the pointer, so we count that. Essentially, if you just increment the counter upon entering the recursive find method, you will get the correct results.

1. **Question: I'm confused about the two parameters in the constructor:**

**ObjectAllocator \*OA = 0**

**bool ShareOA = false**

**They seem to contradict each other.**

**Answer:** The parameters are not related.

The first one is used to provide an allocator to the BSTree/AVLTree object. There are two possible situations:

* + If the parameter is non-NULL, then a **valid** allocator has been provided. The BSTree object should use that allocator for all of its allocations. Also, the object does not own the allocator and should not destroy (free) it. The owner of the allocator (the driver) will free it when it's finished with it.
  + If the parameter is NULL, then the BSTree object must create its own allocator and use that for all of its allocations. The object now owns the allocator and must destroy (free) it when the object is finished with it (most likely in the destructor). Make sure to set *UseCPPMemManager* to **true** when you construct your own ObjectAllocator. This will make sure that the allocator will have enough memory (pages) for the client.

The second parameter tells the copy constructor and assignment operator how to construct the copy or do the assignment (i.e. whether or not to use the right-hand-side allocator).

1. If the object being copied/assigned from has its SharedAllocator set to **true**, then the copy will use the allocator from the copied (right-hand-side) object. The object does not own the allocator and should not destroy (free) it. The value of SharedAllocator should be propagated to other copies made from this object. This allows one ObjectAllocator to be used for all BSTrees in the program.
2. If the object being copied has its SharedAllocator set to **false**, then the copy will create its own allocator and use that for all of its allocations. The object now owns the allocator and must destroy (free) it when the object is finished with it (most likely in the destructor). This means that the copy **will not** share its allocator with other copies.

Some sample code should clarify this:

**template** <**typename** T>

BSTree<T>::BSTree(**const** BSTree &rhs)

{

*// If we are to use rhs' allocator*

**if** (rhs.shareAllocator\_)

{

objAllocator\_ = rhs.objAllocator\_; *// Use rhs' allocator*

freeAllocator\_ = **false**; *// We don't own it (won't free it)*

shareAllocator\_ = **true**; *// If a copy of 'this object' is made, share the allocator*

}

**else** *// No sharing, create our own personal allocator*

{

OAConfig config(**true**); *// Set UseCPPMemManager to true, default the others*

objAllocator\_ = **new** ObjectAllocator(**sizeof**(BinTreeNode), config);

freeAllocator\_ = **true**; *// We own the allocator, we will have to free it*

shareAllocator\_ = **false**; *// Do not share this allocator with any other list*

}

*// other stuff*

}

Note: Don't copy the value of SharedAllocator or Allocator in the copy constructor or assignment operator. Also, DO NOT make a copy of the ObjectAllocator in your copy constructor!

1. **Question: I don't see a copy constructor or assignment operator in the AVLTree class. Is that a**
2. **mistake?**

**Answer:** No. Since there is no additional data in AVLTree that needs to be copied or assigned, the default copy constructor and assignment operator generated by the compiler are adequate. If you do add additional fields to the AVLTree's private section, you *may* need to define and implement these methods for them to work properly. Consult your CS170 notes regarding when the compiler-generated methods are inadequate.