

PONDER 09 : BINARY SORT

Due Saturday at 11:59 PM MST

The next programming assignment will be to implement the binary search tree data structure and use it to implement the binary sort.

Binary Search Tree

Create a class to capture the notion of a binary search tree. This will encapsulate the Binary Node, so the clients of the class will have no knowledge or access to any of the implementation details. Of course, any data type will need to be supported, so your class will be a template class. It will need to be defined in its own header file (bst.h). The class name must be BST and will need to support the following operations:

* **Constructor**: Default constructor setting root to NULL, and the copy constructor. In the case of an allocation error, the following c-string exception will be thrown:  
  ERROR: Unable to allocate a node
* **Destructor**: Delete all the elements in the binary tree, probably calling the functiondeleteBinaryTree() from the previous lesson.
* **operator=**: Assignment operator. Copy one binary search tree onto another.
* **empty()**: Determines whether the current binary search tree is empty. This can be accomplished by testing if the root node is NULL.
* **size()**: Return the number of nodes in the binary search tree. This method takes no parameters and returns an integer value.
* **clear()**: Delete all the elements in the binary tree.
* **insert()**: The parameter is the data item to be inserted into the binary search tree. This will involve finding the appropriate location in the tree for the data item and creating a node to encapsulate it. There is no return value. In the case of an allocation error, the following c-string exception will be thrown:  
  ERROR: Unable to allocate a node
* **remove()**: The parameter is an iterator referring to the item to be removed from the binary search tree. The node will then be removed following the algorithm presented in the textbook (titled "Removing a Node from a BST" in chapter 12.4). There is no return value.
* **find()**: The parameter is the data item to be found and the return value is an iterator referring to the element in the tree corresponding to the parameter. If no element is found, the end() iterator will be returned.
* **begin()**: There is no parameter, and the return value is an iterator referring to the leftmost element in the BST.
* **end()**: There is no parameter, and the return value is an iterator referring to no element in the BST. By convention, this is the NULL iterator.
* **rbegin()**: There is no parameter, and the return value is an iterator referring to the rightmost element in the BST.
* **rend()**: There is no parameter, and the return value is an iterator referring to no element in the BST. By convention, this is the NULL iterator.

In addition to the BST class, it is necessary to have a BSTIterator class which will iterate through the binary search tree. The BSTIterator class is provided in bst.h

Driver Program

A driver program is provided. This file (/home/cs235/week09\_E/week09.cpp) will pound-include your header file (bst.h) and expect a template class BST to be defined therein. I have provided a completed version of the BST Iterator class in "/home/cs235/week09\_E/bst.h" You may use this file but you must complete rest of the BST class functions. The BSTIterator class is complete.   
This program will exercise your class, filling the container with user input and displaying the results. As with previous assignments, a makefile will be provided (/home/cs235/week09\_E/makefile). You will need to use the following header files: sortBinary.h,bnode.h, and bst.h.

Binary Sort

In addition to passing the four test functions for the BST class, you will also need to use the BSTclass to implement the binary sort. Perhaps this will be the simplest part of the program if the BSTand BSTIterator classes were correctly implemented. This function will take a template array as a parameter as well as the number of items in the array. It will then return the sorted array. To do this, increment through the array and insert each item into a binary search tree. Next, iterate through the binary search tree and put each element back into the array.

A few hints to help you with your program:

* Draw a small binary search tree on a sheet of paper. The ones in testAdd() andtestIterate() are good places to start. Next, write code similar to what we used in the previous lesson (Week 08) to ensure the tree is shaped the way you expect.
* When writing the increment operator for BSTIterator, create a special tree representing single increment cases as described above (for example: an empty stack, when there is just a single right child, or when the parent is the root). Do not start with an overly complex tree.

Common Mistakes

The most common mistakes students make with this assignment include the following:

* **Incorrect insert() function**. If the insert() function does not work, then everything else about the tree will fail to work also.
* **Segmentation fault**. Usually this occurs when you attempt to dereference a pRight orpLeft pointer that is set to NULL. Make sure to do a NULL check before you dereference anything!

Test Bed

The testBed for this assignment is:

testBed cs235/week09 week09.tar

You can also run testBed on the executable:

testBed cs235/week09 a.out

Of course, you will need to pass testBed to get full credit on the assignment.

Submitting

You will submit this assignment individually using the Linux submit command. Please:

1. Create a TAR file built from the makefile, which will contain several files:
   * makefile: Directly from /home/cs235/week09/makefile except with your edits on the comment block.
   * bnode.h: Your class definition for BinaryNode.
   * bst.h: Your class definition for BST and BSTIterator.
   * stack.h: Your class definition for Stack created from an earlier assignment.
   * sortBinary.h: The template function for sortBinary().
   * week09.cpp: Unmodified from /home/cs235/week09/week09.cpp.
2. Run the program by hand a few times through all four test cases as well as the binary sort algorithm.
3. Verify your solution with testBed.
4. Submit your file using the submit command. The submit command will prompt you for your instructor, the class (cs235), and the assignment (week09). You submit your file with:

submit week09.tar

Your program will be graded according to the following rubric:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Exceptional 100% | Good 90% | Acceptable 70% | Developing 50% | Missing 0% |
| BST interface  10% | The interfaces are perfectly specified with respect to const, pass-by-reference, etc. | week09.cppcompiles without modification | All of the methods in BST match the problem definition | BST has many of the same interfaces as the problem definition | The public methods and variables in theBST class do not resemble the problem definition |
| BST Implementation  20% | Passes all fourBST testBed tests | Passes three testBed tests | Passes two testBed tests | Passes one testBed test | Program fails to compile or does not pass any testBed tests |
| BSTIterator  30% | Solution works, is elegant, and efficient | Both forward and reverse iterators work | Works in some limited cases | Elements of the solution are present | No attempt was made to iterate through the BST |
| Binary Search  10% | The code is elegant and efficient | Passes the Binary Search testBed test | The code essentially works but with minor defects | Elements of the solution are present | The Binary Search problem was not attempted |
| Code Quality  20% | There is no obvious room for improvement | All the principles of encapsulation and modularization are honored | One function is written in a "backwards" way or could be improved | Two or more functions appears "thrown together" | The code appears to be written without any obvious forethought |
| Style  10% | Great variable names, no errors, great comments | No obvious style errors | A few minor style errors: non-standard spacing, poor variable names, missing comments, etc. | Overly generic variable names, misleading comments, or other gross style errors | No knowledge of the BYU-I code style guidelines were demonstrated |

Please make sure to fill out the program header in the makefile.