Lab 08

Resources

- Discussion slides (./lab08slides.pdf) - Copy of the presentation given by the TA
- Nemo's Discussion on Data Structures (http://www.cise.ufl.edu/~nemo/cop3503/slides/Lecture+20_data_struct.pptx) some information on Binary Search Trees, which you will be implementing
- main.cpp (./main.cpp) - The main I will be using to test.
- out.txt (./out.txt) - Example of a successful output.

Idea

This lab will build upon the BST we made in Lab 07. It will get you familiar with dyna memory management and pointers.

In-lab assignment

Description

You will be adding functionality into your BST class. If you do not have a functioning class, you can create that one for this lab. I mentioned last week that your BSTs have memory leaks (because you are using the new keyword). Your main job for this lab is that. We will also be doing some more work with traversals.

Requirements/Deliverables

Lab 07 Stuff

You will implement a standard BST. The tree will be made up of many Nodes, which y will also need to implement. Make the tree function as closely to the way described in you can. You will need to break up your implementation into .h and .cpp files as usual

!!!!!! IT IS VERY IMPORTANT THAT YOU NAME THESE CLASSES AND FUNCTION EXACTLY AS THEY APPEAR HERE !!!!!!

The tree class should be called BST and needs to implement the following functions (t are prototypes):

1.
```
BST()
```

   This is the default constructor. It should initialize the root node to be nullptr.

2.
```
void insert(int)
```

   This function inserts a given value into the tree. YOU MUST KEEP THE ORDER (THE TREE. You keep tree order by placing nodes less than the root to the left anc nodes greater than the root to the right. If you are given a duplicate node, you car simply ignore it (don't add it to the tree). I will not give you duplicate nodes in m testing.
   - For reference, the tree below was built by inserting nodes in this order: 2 -> ; -> 5 -> 4

3.
```
void print_inorder()
```

   This is an inorder traversal of your tree. It should print the nodes of the tree in so order. The format should be each value separated by a single space on one line. T should be a new line at the end. Below is an example

```
  2
 / \
1   3
     \
      5
     /
    4
```

The result of the print function is:

```
1 2 3 4 5 \n
```

If the tree is empty, just print "Tree empty".

4. ```
   bool find(int)
   ```

   This function returns true if the given value is in the tree, false otherwise.

5. ```
   void print_from_value(int)
   ```

   This function searches for the value in the tree and prints the subtree rooted at tl node using an inorder traversal. Taking the tree from above, calling this function passing 5 should print

   ```
   4 5 \n
   ```

   If the node is not in the tree, print a message saying "Node [whatever number wa passed] not found".

You may want to add in additional functions to accomplish intermediate taks. These functions are helper functions and should be *private*.

The class for the node should be called Node. I will not specify the functions that shou in the node class. *hint hint, the node functions should be the ones that are recursive, you will probably need functions very similar to those found in the BST class*. For instance, the BST::print_inorder function can be written as

```
void print_inorder(){
    if(root == nullptr){
        cout << "Tree empty" << endl;
        return;
    }
    root->print_inorder();
}
```

if you write the Node::print_inorder function correctly.

New Additions

Assuming you have all the above things working, you must now add in the following functionality (again, your BST functions must have the same signature that appear he

1. ```
   void remove(int)
   ```

   This function will remove a given value from the BST. If the value is not in the tre nothing should be done (No need to print an error message). If your tree allows duplicates, you can decide if you want to remove all or just one of the values. Foll procedure discussed in lab to accomplish this. Do not forget to free the node that removed. AVOID MEMORY LEAKS. This function does not need to be recursive.

2. ```
   void clear()
   ```

   This function will clear ALL nodes from your tree. After calling this, you tree shou completely empty (like it was just constructed). AVOID MEMORY LEAKS.

3. ```
   ~BST()
   ```

   This is the destructor for BST. It should free any memory that was dynamically allocated in the BST. HINT HINT, if only you had already written some nifty way *clear* the tree of all its nodes, this function would be very easy.

4. ```
   int sum()
   ```

This function returns the sum of all the nodes in the tree. An empty tree should re
0. In order to get this value, you will have to *traverse* through everything single n
the tree, keeping a sum total.

5.
```
int size()
```

This function returns the current number of nodes in your tree. It should be very similar to the sum() function.

6.
```
float average()
```

This function returns the average value of all the nodes. HINT HINT: The average defined as (sum of all the nodes)/(total number of nodes). If only you had some w calculate the sum and total, you could get the average in one line... Be careful of ty conversions. Ex: 4/3 returns 1 in c++. 4.0/3 returns 1.333333...

Submission

You will only need to submit a BST.h and a BST.cpp file containing your implementat the assignment Lab 08 area of elearning. Just submit them as two separate attachme no need to zip them up. You can put both the Node and BST classes in the same files (need for a node.h and node.cpp). *There should not be a main() in either of these files!* will write my own main.cpp to test your BST class using the functions described. The file I will use to grade can be found here (./main.cpp). If you finish during lab time, come by and check you off on this test program. Otherwise, you have until Sunday nig turn in the program.

Hints

- Again, all of the recursion will be done in the Node class. Your BST functions shou very simple.
- You will probably want to write your own test harness main to test your program you go instead of trying to go for the one used for grading directly.
- Make sure you are initializing your pointers correctly (to nullptr or otherwise).
- Don't forget about the debugger lesson from last (last last?) week. Use it early anc it often and you will get more and more profiicient at it.

- There isn't an easy way to check if you are actually freeing all of your memory. On you can keep tabs is to put a print statement in you destructors (you can write on ~Node() as well) to make sure it gets called on every value.

Grading Distribution

- 5 points for passing the tests in the given main
- 5 points for the delete node function (1 for a leaf node, 2 for a node with only 1 chi for a node with 2 children [1 for that node being the root, 1 for that node being ar other node])
- 5 points for the clear function
- 5 points for the sum/size/average functions

Optional Enhancements

- There is no need for the payload of these nodes to be integers. This is just an arbi choice. Try editing your functionality to store Strings, People, or anything you ca think of (templates could be a good idea here). HINT: It would make more sense t store a pointer to the data then. DOUBLE HINT: How does this change the destru for the node?
- There is a concept often applied to trees called *balancing*. A balanced tree is one which nodes in each level have the maximum amount of children (so ideally, each has 2 children). This avoids the tree degenerating into a linked list. First of all, wh this a bad thing? Secondly, how can you balance a tree? There are many ways. So keep the tree balanced as you add nodes, and some take an unbalanced tree and r it balanced. You can look up red/black trees or AVL trees for the former and the I Stout-Warren algorithm is a good place to get started for the latter.

```
Unbalanced Tree
      1
     / \
    0   5
       / \
      2   6
           \
            7
             \
              8

Same Tree But Balanced
            5
          /     \
         1       7
        / \     / \
      0   2   6   8
```