Assignment-2

Binary Tree Class Implementation

For this assignment, you are asked to complete a C++ implementation of a general binary tree class. This is *not* specifically a binary search tree (that will be your assignment 3). The class is named BinaryTree, and is implemented as a template class where the class parameter is the type of the element stored in each node. **See BinaryTree.h and BinaryTree.cpp for p**artial implementations. There is also a short **main** function **in main.cc**, that illustrates construction and display. (See also the Testing section below.)

You will also need PDF.h, PDF.cc and PDFFonts.cc to get the graphical PDF output. All files are provided in the zip folder.

Structure for Tree Nodes

The BinaryTree class uses the following struct to represent nodes.

```
template <class T>
struct BTNode {
  T       elem;  // element contained in the node
  BTNode *left;  // pointer to the left child (can be NULL)
  BTNode *right; // pointer to the right child (can be NULL)

  // Constructors
  BTNode();
  BTNode( T elem, BTNode* left = NULL, BTNode* right = NULL );
  BTNode( const BTNode& src );

  // Simple tests
  bool is_leaf() const;
};
```

Binary Tree Class

Your BinaryTree class is implemented as a template class, where the class parameter T is the type of the element stored in a node (i.e., it matches the T class parameter in BTNode). The following is a template for the BinaryTree class. Note that the only data member is a pointer to a BTNode, the root. (The access for the root node is protected, which means is will be visible to subclasses but otherwise is private.)

```
template <class T>
class BinaryTree {
 public:
```

```
  /* Construction */
  BinaryTree() { root = NULL; }
  ...

 protected:
  BTNode<T> *root;  // Root node (NULL if the tree is empty)
  ...
};
```

Constructors

- BinaryTree() – constructs an empty binary tree.
- BinaryTree(T *elements, int n_elements) – constructs a complete tree having elements, in the usual order of a complete binary tree.
  Note: the code implementing this is provided.
- BinaryTree(const BinaryTree& src) – copy constructor; constructs a duplicate of src
- ~BinaryTree() – destructor (deletes all the nodes in this tree).

Access and Tests

- bool is_empty() – returns true if this tree is empty, and false otherwise
- int node_count() const – returns the total number of nodes in this tree
- int leaf_count() const – returns the number of leaf in this tree
- int height() const – returns the height of this tree: the height of the empty tree is 0; the height of a nonempty tree is one plus the longest path from the root to any leaf.

Mutators, and other Initialization

- bool empty_this() – empties (and deallocates) this tree
- int to_flat_array(T* nodes, int max) const – copies the node elements to the nodes array assuming this is a complete binary tree.
  Note: the code implementing this is provided.

Traversal

- void preorder(void (*f)(const T&)) const – Performs a preorder traversal of this tree, applying f on the element of each node visited.
- void inorder(void (*f)(const T&)) const – Performs an inorder traversal of this tree, applying f on the element of each node visited.
- void postorder(void (*f)(const T&)) const – Performs a postorder traversal of this tree, applying f on the element of each node visited.

Operators

- bool operator==(const BinaryTree& src) const – returns true if this and src are identical; that is, if the tree structures are the same and the corresponding elements match.

- bool operator!=(const BinaryTree& src) const – logical complement of the == operator.
- BinaryTree& operator=(const BinaryTree& src) const – assignment operator: copies src to this tree (removing the prior content).

Input/Output

- template<class S> friend ostream& operator<<(ostream& out, BinaryTree<S>& src); – writes the sequence of nodes of src in complete-tree order to out assuming this is a complete binary tree.
  Note: the code implementing this has been provided.

Compilation

The command-line compilation command (which will be used to compile and test your submitted code) is
g++ PDF.cc PDFFonts.cc main.cc
(or with test.cc in place of main.cc).

Testing

As usual, you are responsible for testing your own code. The file test.cc, which has its own main() function, contains some of the tests that will be used for your final submission. It is important to check if tree is balanced or complete after adding or deleting symbols. You will loose points, if your true is left or right dominant. You should implement a function that balances the tree after these operations.

Submission

Please submit your BinaryTree.h and BinaryTree.cpp files along with report.
Report
You have to explain each and every functions, including binarytree.cpp, binarytree.h, pdf.cc, pdffonts.cc, main.cc and pdf.h. Also explain how in-order, post-order, pre-order works with examples.

Few Clarifications

1. *In developing your methods, do not assume that the binary tree is complete. Your tree can be complete, full or perfect. An ideal implementation will start with perfect tree and then move towards complete tree. Modify class to satisfy these conditions. Please refer to slides and theorem to better understand these concepts*
2. *The display method uses the height method to determine the scale of the output. So if your height method isn't right, the output may be too large or too small.*
3. *You can add functionality to the BinaryTree.h and BinaryTree.cpp, such as helper functions for the == and = operators.*
4. *Start Early, and if you have any doubts simply email (cc your TA's in all your communication)*