Binary Search Tree

You are asked to implement a template Binary Search Tree. As part of your implementation you will be required to read data from a file and store it in the your tree.

The items stored in your BST are objects, e.g.

```
struct INT {
        // data member
        int *_data;
        /* various member functions including
         *  constructor, copy constructor, operator==(), operator=(), operator<(),
        *  destructor, etc.
        */
};

struct CHAR{
// data member
        char *_data;
        /* various member functions including
         *  constructor, copy constructor, operator==(), operator=(), operator<(),
        *  destructor, etc.
        */

};

template<typename T>
class BNode{
        // data members
        T *_node;
        BNode *_left;
        BNode *_right;
        /* various member functions including
         *  constructor, copy constructor,  operator=(), operator<(),
        *  destructor, etc.
        */
};

template<typename TreeType>
class BinaryTree{
        BNode<TreeType> *_root;
        size_t _nodeCount;
        /*
                pintInorderTraversal(),printPostorderTraversal(),
                printPreorderTraversal(), find(), deleteAll(), insert(), constructor,
                destructor, height(), delete()
      */
};
```

Public Interfaces that will be tested

Interface	**Comment**	**Tree Height(2)**	**Tree Height(3)**	**Tree Height(>4)**
insert(Type)	inserts object Type into tree{Debug message: if Type is a duplicate, print out message to that effect.}			
height(Type)	Return height of node that contains key. if invoked with no arguments, returns height of tree			
printInOrder()	print tree using inoder traversal			
printPostOrder()	print tree using postorder traversal			
printPreOrder()	print tree using preorder traversal			
find(Type)	return **true** if value of object Type is in tree, false otherwise			
delete(Type)	delete object with value of Type from tree {debug message: print if object was deleted or not found}			
deleteAll()	delete all objects in tree			
Load(filename)	data file will contain space separated values. bulk loading of BST from datafile			
memory management	You must manage your memory allocation/deallocation. Failure to do so may result in a failing score			

```
. . .
       // short example showing how to construct and print a BST
      // This is just an manual example for self testing !!!!
       std::cout <<" Begin with INT Tree\n" <<std::endl;

       BinaryTree<INT> iTree;
       INT i1(10);
       INT i2(20);
       INT i3(30);
       INT i4(40);
       INT i5(50);
       INT i6(60);
       INT i7(100);

       iTree.insert(i4);
       iTree.insert(i2);
       iTree.insert(i3);
       iTree.insert(i6);
       iTree.insert(i5);
       iTree.insert(i1);
       iTree.printInorderTraversal();

       if(iTree.find(i1))
                std::cout << “found: “ << i1 << std:endl;
       else
                std::cout << “not found: “ << i1 << std:endl;

       if(iTree.find(i7))
                std::cout << “found: “ << i7 << std:endl;
       else
                std::cout << “not found: “ << i7 << std:endl;

       std::cout <<" END with INT Tree\n\n" <<std::endl;

// Not showing reading from file

std::cout <<" Begin with CHAR Tree\n" <<std::endl;
       BinaryTree<CHAR> cTree;

       CHAR v1("abc");
       CHAR v2("bcd");
       CHAR v3("efg");

       // Should support both methods of insertion
       cTree.insert(v2);
       cTree.insert(v3);
       cTree.insert(v1);
       cTree.insert(CHAR("bob"));
       cTree.insert(CHAR("tom"));

       cTree.printInorderTraversal();
       std::cout <<" Finish with CHAR Tree\n\n" <<std::endl;
       cTree.deleteAll();


. . .
```

Notes: pointers to const references

Assume you have a pointer and you pass the pointer to a function to change what the pointer points too. You might try the follow:

```
/*
Simple example to show how to change what a pointer points to
using a function. Must use * &, otherwise the change is only
valid in the function that is changing what the pointer points to.

J. Montgomery
*/
#include <iostream>
using namespace std;
struct hnode{
	int data;
	hnode *ptr;
};

// does not work....
// works with in the function, but change does not persist after
// function returns!!!!
void linkNewWithPointer(hnode *myP, int newData)
{
	hnode *t = new hnode;
	t->data = newData;
	t->ptr = NULL;

	//assign myP to newly allocated node
	myP = t;

	cout << myP->data << endl;
	return;

	// this also results in a memory leak, becuase we have no
	// way to refer to memory allocated above!!!
}

// Changing what a pointer points to persists in this version
// note * &, (this is a pointer to reference....
void linkeNewWithPointerReference(hnode* &myP, int newData)
{
	hnode *t = new hnode;
	t->data = newData;
	t->ptr = NULL;

	//assign myP to newly allocated node
	myP = t;


	cout << myP->data << endl;
	return;
	// no memory leak! the change to what myP points to persist (myP is
_node->ptr from main())
}
```

```
int main(int argc, char argv[]){

	hnode *_root = NULL;

	_root = new hnode; // o.k.
	_root->data = 20;
	_root->ptr = NULL; // _root points to new created hnode.

	//new use method/function to add a new hnode and link it to _root
	linkNewWithPointer(_root->ptr, 30);
	//_root->ptr should point to new allocated hnode, it does not.
	// The code will cause an exception....

	 //cout <<" Tried to use function linkNewWithPointer() to attach a
	 // new hnode to _root->ptr: (value should be 30)" << _root->ptr->data << endl;

	// try it with a pointer to a reference
	// changing what the pointer points to persist after function call!!!
	linkeNewWithPointerReference(_root->ptr, 30);
	//_root->ptr->data now exists...
		cout <<" Tried to use function linkNewWithPointerRefernece() to"
		cout <<" attach a new hnode to _root->ptr: (value should be 30)"
			<< _root->ptr->data << endl;

	delete _root->ptr;  // delete child
	delete _root;        // delete parent
	return 0;
}
```