# Binary Search Tree (Part 1):

***(60 pts, Due Thursday, Oct 29)***

***Note: This is the first part of a 2-part lab. Only this part is due on Oct 29. But please get it working or you will struggle with the second part.***

***For the second part, the data type will be changing. For now, it is a string.***

## Contents

# Explanation:

For this lab you will be writing the basic methods for a binary search tree. The data in the binary search tree is of type Phrase (a class I am giving to you), and, while this lab is due in 1 week, be aware that many of the methods in this lab will be re-used in the lab that follows it.

To visualize the binary search tree as you are working on it, you can draw it, or you can use numerous on-line visualizers, including this one:

https://www.cs.usfca.edu/~galles/visualization/BST.html

At the end of this document I have included the output from my lab.

The code I am giving you is:

- Phrase.hpp and Phrase.cpp
- TNode.hpp and TNode.cpp
- bstmain.cpp
- BST.hpp

You must write the methods in BST.cpp that are specified (below):

# BST.cpp:

## Code you must write:

The code you must write: BST.cpp (i.e., the methods associated with BST.hpp, right below this section, although I am giving you some of the methods in there as well)

```
/*BST.cpp: You must write the BST.cpp and complete the following methods:*/
```

bool insert(string s) (7): `this method takes as an input parameter a string (which will go into the phrase field of the data when a node is created to be inserted) and returns true if the data is inserted successfully, false otherwise.`
`Be aware: when you insert a new node into the binary search tree, this method should call the setHeights method to adjust the heights`

TNode *find(string s) (4):`finds whether s is in the phrase part of the data in the tree,and, if it is, returns the node holding s. Otherwise it returns NULL.`

void printTreeIO(Tnode *n)(3): `recursive function that prints out the data in the tree in order`

void printTreePre(Tnode *n) (3): `a recursive function that prints out the datain the tree in pre-order`

void printTreePost(Tnode *n) (3): a recursive function that prints out the data in the tree in post-order

TNode *remove(string s)(8) – this method removes a node from the tree, and returns that node. There are 3 cases when you remove a node: either the node being removed has no children (left or right), in which case this method calls the method removeNoKids, the node you are removing has only one child, in which case the method calls removeOneKid (with a Boolean flag set to true if there is a left child, and false if there is a right child), or the node being removed has both a left and a right child, in which you replace the node being removed with the appropriate replacement child, and then remove the node used as a replacement by calling either removeNoKids or removeOneKid, depending on which is appropriate.
NOTE: I used the rightmost of the left child as a replacement. To get my output, you must do the same.

TNode *removeNoKids(TNode *tmp)(5): for removing a node with no children

TNode *removeOneKid(TNode *tmp, bool leftFlag)(7): for removing a node with one child, with the leftFlag indicating whether the node's child is either the left child or the right child.

void setHeight(TNode *n)(10): This method sets the heights of the nodes in a tree. Once a node is inserted, only the node's ancestors can have their height changed. Thus you should set the height of the node being inserted (to 1) and then adjust the heights of the node's parent, grandparent, etc. up until either the height of the node doesn't change or you hit the root.

20 pts for getting everything to work together

## Methods in CPP I'm Giving You:

```
BST::BST() {
	root = NULL;
}

BST::BST(string s) {
	root = new TNode(s);
}

void BST::clearTree() {
	if (root == NULL) {
		cout << "Tree already empty" << endl;
	}
	else {
		cout << endl << "Clearing Tree:" << endl;
		clearTree(root);
		root = NULL;
	}
}

void BST::clearTree(TNode *tmp) {
	if (tmp == NULL) {
```

```
            return;
        }
        else {
            clearTree(tmp->left);
            clearTree(tmp->right);
            tmp->printNode();
            delete(tmp);
        }
}

/*Note: the following three functions' sole job is to call printTreeIO(TNode
*t),printTreePre(TNode *t), and printTreePost(TNode *t) while printint out which
Function is being called.
YOU MUST STILL WRITE THE RECURSIVE VERSION OF THESE FUNCTIONS!!!*/
void BST::printTreeIO() {  // Just the start – you must write the recursive version
        if (root == NULL ) {
            cout << "Empty Tree" << endl;
        }
        else {
            cout << endl<<"Printing In Order:" <<endl;
            printTreeIO(root);
        }
}
void BST::printTreePre() {
        if (root == NULL ) {
            cout << "Empty Tree" << endl;
        }
        else {
            cout << endl<<"Printing PreOrder:" <<endl;
            printTreePre(root);
        }
}
void BST::printTreePost() {
        if (root == NULL ) {
            cout << "Empty Tree" << endl;
        }
        else {
            cout << endl<<"Printing PostOrder:" <<endl;
            printTreePost(root);
        }
}
```

## BST.HPP (Giving You):

Here is the accompanying BST.hpp code:

```
#ifndef BST_HPP_
#define BST_HPP_

#include "TNode.hpp"

class BST {
        TNode *root;
```

```
public:
	BST();
	BST(string s);
	bool insert(string s);
	TNode *find(string s);
	void printTreeIO();
	void printTreeIO(TNode *n);
	void printTreePre();
	void printTreePre(TNode *n);
	void printTreePost();
	void printTreePost(TNode *n);
	void clearTree();
	void clearTree(TNode *tmp);
	TNode *remove(string s);
	TNode *removeNoKids(TNode *tmp);
	TNode *removeOneKid(TNode *tmp, bool leftFlag);
	void setHeight(TNode *n);
};
#endif /* BST_HPP_ */
```

```
################################################################################
```

/*Phrase.hpp*/

```
#ifndef PHRASE_HPP_
#define PHRASE_HPP_

#include <iostream>
using namespace std;

class Phrase{
	friend class TNode;
	friend class BST;
	string phrase;
public:
	Phrase(string s);
	Phrase();
};

#endif /* PHRASE_HPP_ */
```

/*Phrase.cpp*/

```
#include <iostream>
#include <string>
#include "Phrase.hpp"
using namespace std;

Phrase::Phrase(string s) {
	phrase = s;
}

Phrase::Phrase() {
```

```
	phrase = "";
}
################################################################################
```

/*TNode.hpp*/

```
#ifndef TNODE_HPP_
#define TNODE_HPP_
#include <iostream>
#include "Phrase.hpp"
using namespace std;

class TNode{
	friend class BST;
	TNode *left;
	TNode *right;
	TNode *parent;
	Phrase *data;
	int height;
public:
	TNode(string s);
	TNode();
	~TNode();
	void printNode();
};

#endif /* TNODE_HPP_ */
```

/*TNode.cpp*/

```
#include <iostream>
#include <string>
#include "TNode.hpp"
using namespace std;

TNode::TNode(string s) {
	left = NULL;
	right = NULL;
	parent = NULL;
	height = 1;
	data = new Phrase(s);
}
TNode::TNode() {
	left = NULL;
	right = NULL;
	parent = NULL;
	height = 1;
	data = new Phrase();
}
TNode::~TNode(){
	cout <<"Deleting "<<data->phrase<<endl;
}
void TNode::printNode() {
	cout << data->phrase<<","<<height<<endl;
}
```

################################################################################

/*bstmain.cpp*/

```
#include "BST.hpp"
#include <iostream>
using namespace std;

int main() {
	string arr[] = {"e","g","f","a","c","d","b"};
	BST *tree = new BST();
	for (int i = 0; i < 7; i++) {
		cout << arr[i]<<", ";
		tree->insert(arr[i]);
	}
	cout << endl;
	tree->printTreePre();
	tree->printTreeIO();
	tree->printTreePost();
	tree->clearTree();
	cout <<"**********************************" << endl;
	string arr3[] =
{"i","was","contemplating","the","immortal","words","of","socrates","who","said,","i"
,"drank","what"};
	for (int i = 0; i < 13; i++) {
		cout << arr3[i]<<", ";
		tree->insert(arr3[i]);
	}
	cout << endl;
	tree->printTreePre();
	tree->printTreeIO();
	tree->printTreePost();
	cout << endl<<"REMOVING DRANK" << endl;
	tree->remove("drank");
	tree->printTreePre();
	tree->printTreeIO();
	tree->printTreePost();
	cout << endl<<"REMOVING IMMORTAL" << endl;
	tree->remove("immortal");
	tree->printTreePre();
	tree->printTreeIO();
	tree->printTreePost();
	cout << endl<<"REMOVING WAS" << endl;
	tree->remove("was");
	tree->printTreePre();
	tree->printTreeIO();
	tree->printTreePost();
	cout << endl<<"REMOVING I" << endl;
	tree->remove("i");
	tree->printTreePre();
	tree->printTreeIO();
	tree->printTreePost();
	tree->clearTree();
	return 0;
}
```

##########################################################################################

## Output:

```
e, g, f, a, c, d, b,

Printing PreOrder:
e,4
a,3
c,2
b,1
d,1
g,2
f,1

Printing In Order:
a,3
b,1
c,2
d,1
e,4
f,1
g,2

Printing PostOrder:
b,1
d,1
c,2
a,3
f,1
g,2
e,4

Clearing Tree:
b,1
Deleting b
d,1
Deleting d
c,2
Deleting c
a,3
Deleting a
f,1
Deleting f
g,2
Deleting g
e,4
Deleting e
**********************************
i, was, contemplating, the, immortal, words, of, socrates, who, said,, i, drank, what,

Printing PreOrder:
i,7
contemplating,2
drank,1
was,6
the,5
immortal,4
of,3
socrates,2
said,,1
```

```
words,3
who,2
what,1

Printing In Order:
contemplating,2
drank,1
i,7
immortal,4
of,3
said,,1
socrates,2
the,5
was,6
what,1
who,2
words,3

Printing PostOrder:
drank,1
contemplating,2
said,,1
socrates,2
of,3
immortal,4
the,5
what,1
who,2
words,3
was,6
i,7

REMOVING DRANK

Printing PreOrder:
i,7
contemplating,1
was,6
the,5
immortal,4
of,3
socrates,2
said,,1
words,3
who,2
what,1

Printing In Order:
contemplating,1
i,7
immortal,4
of,3
said,,1
socrates,2
the,5
was,6
what,1
who,2
words,3

Printing PostOrder:
```

```
contemplating,1
said,,1
socrates,2
of,3
immortal,4
the,5
what,1
who,2
words,3
was,6
i,7

REMOVING IMMORTAL

Printing PreOrder:
i,6
contemplating,1
was,5
the,4
of,3
socrates,2
said,,1
words,3
who,2
what,1

Printing In Order:
contemplating,1
i,6
of,3
said,,1
socrates,2
the,4
was,5
what,1
who,2
words,3

Printing PostOrder:
contemplating,1
said,,1
socrates,2
of,3
the,4
what,1
who,2
words,3
was,5
i,6

REMOVING WAS

Printing PreOrder:
i,5
contemplating,1
the,4
of,3
socrates,2
said,,1
words,3
who,2
```

```
what,1

Printing In Order:
contemplating,1
i,5
of,3
said,,1
socrates,2
the,4
what,1
who,2
words,3

Printing PostOrder:
contemplating,1
said,,1
socrates,2
of,3
what,1
who,2
words,3
the,4
i,5

REMOVING I

Printing PreOrder:
contemplating,5
the,4
of,3
socrates,2
said,,1
words,3
who,2
what,1

Printing In Order:
contemplating,5
of,3
said,,1
socrates,2
the,4
what,1
who,2
words,3

Printing PostOrder:
said,,1
socrates,2
of,3
what,1
who,2
words,3
the,4
contemplating,5

Clearing Tree:
said,,1
Deleting said,
socrates,2
Deleting socrates
```

```
of,3
Deleting of
what,1
Deleting what
who,2
Deleting who
words,3
Deleting words
the,4
Deleting the
contemplating,5
Deleting contemplating
```