# Let's practice our traversals.

## Objectives

* Build an ADT of a a Binary Search Tree using a node containing a left pointer and a right pointer.
* Use that Binary Search Tree to perform traversals.

## Instructions

Write a program which...

* Prompt the user to enter the number of values that they would like to insert into a binary search tree.
* Prompt the user to insert, one-at-a-time, a value into the binary search tree.
* Print the preorder, postorder, and inorder traversal on the tree.

## Binary Search Tree

The methods preorder, postorder, inorder, preorderR, postorderR, and inorderR will all contains the necessary screen output statements to verify that the traversal is happening correctly (see the examples for the correct format of this output).

Hint: The copy and the deleteTree methods are also tree traversals. The copy is a preorder traversal and you'll want to **insert** (method) a new value into a based on an existing node. The deleteTree method will **delete** (keyword) a node based on a postorder traversal.

#ifndef BSTree\_HPP  
#define BSTree\_HPP  
#include <iostream>  
  
template <class T>  
struct BSTNode {  
 T data;  
 BSTNode<T>\* left;  
 BSTNode<T>\* right;  
};  
  
class BSTree {  
private:  
 int mSize; // Records the size of the tree  
 BSTNode\* mRoot; // Maintains the root of the tree  
 void copy(const BSTNode<T>\*); // Copies the tree  
 void deleteTree(BSTNode<T>\*); // Deletes every node in the tree  
  
 void preorderR(const BSTNode<T>\*) const; // Recursive preorder traversal  
 void postorderR(const BSTNode<T>\*) const; // Recursive postorder traversal  
 void inorderR(const BSTNode<T>\*) const; // Recursive inorder traversal  
 int heightR(const BSTNode<T>\*) const; // Recursive height  
  
public:  
 BSTree(); // Init constructor  
 BSTree(const BSTree<T>&); // Copy constructor  
 ~BSTree(); // Destructor  
 BSTree& operator=(const BSTree<T>&); // Assignment constructor  
  
 void clear(); // Clears tree by calling deleteTree  
 // if mRoot node is not null  
 int size() const; // Returns size of tree  
 void insert(int); // Inserts one item into tree  
  
 void preorder() const; // Begins preorder traversal  
 void postorder() const; // Begins postorder traversal  
 void inorder() const; // Begins inorder traversal  
 int height() const; // Returns height of tree  
};  
  
// Implementation goes here.  
  
#endif

## Sample Output

Here's the output with 7 values (5, 3, 7, 2, 4, 6, 8). This is a perfectly balanced tree.

How many values do you wish to insert? 7  
1: 5  
2: 3  
3: 7  
4: 2  
5: 4  
6: 6  
7: 8  
  
Preorder : 5 3 2 4 7 6 8  
Postorder : 2 4 3 6 8 7 5  
Inorder : 2 3 4 5 6 7 8  
Height : 3

Here is an example of the values 1 to 5 in order.

How many values do you wish to insert? 5  
1: 1  
2: 2  
3: 3  
4: 4  
5: 5  
  
Preorder : 1 2 3 4 5  
Postorder : 5 4 3 2 1  
Inorder : 1 2 3 4 5  
Height : 5

Here's Tommy Tutone's "Jenny". This is your typical state with a variety of levels to the tree.

How many values do you wish to insert? 7  
1: 8  
2: 6  
3: 7  
4: 5  
5: 3  
6: 0  
7: 9  
  
Preorder : 8 6 5 3 0 7 9  
Postorder : 0 3 5 7 6 9 8  
Inorder : 0 3 5 6 7 8 9  
Height : 5

Make sure your algorithm works with a tree with no values.

How many values do you wish to insert? 0  
  
Preorder :   
Postorder :   
Inorder :  
Height : 0

## Files

Here is a listing of all files (plus a total line) with the number of lines that I typed in each file (this doesn't include comments). These files should be included in your final submission.

BSTree.hpp, main.cpp