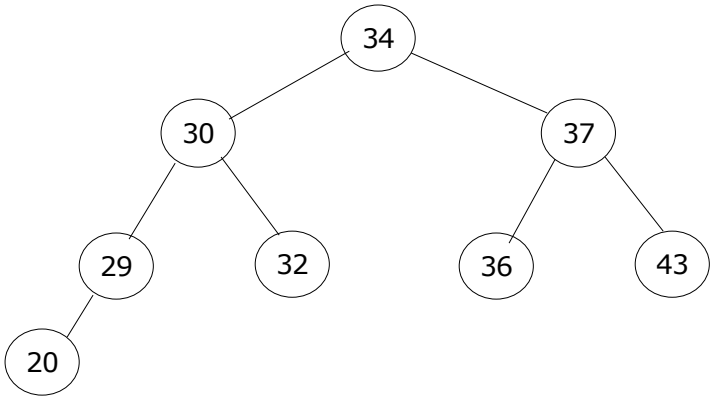
Tree traversals



PreOrder: node, left subtree, right subtree = 34 30 29 20 32 37 36 43
InOrder: left subtree, node, right subtree = 20 29 30 32 34 36 37 43
PostOrder: left subtree, right subtree, node= 20 20 32 30 36 43 37 34
Level Order: level 0, level1, etc. = 34 30 37 29 32 36 43 20

Building a binary search tree

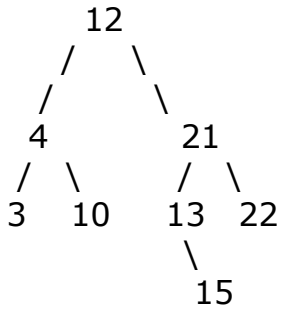
```
#include<iostream>
using namespace std;

class BTNode
{
public:
  int key_value;       // The data in this node.
  BTNode *left;   // Pointer to the left subtree.
  BTNode *right;  // Pointer to the right subtree.
};


class BTree
{
public:
  BTNode* root;
  BTree();   //constructor
  ~BTree(); //destructor
  void insert(int key);       //Takes care of creating root node
  void insert(int key, BTNode *leaf);   //except for the root, this does the insertion
  BTNode *search(int key); //search for key in the whole tree - uses the following function
  BTNode *search(int key, BTNode *leaf); //search for key  beginning here
  void destroy();    //deletes the whole tree, uses the following function
  void destroy(BTNode *leaf); //deletes tree rooted at this node
};

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

separation between Node and Tree makes sense, but not absolutely necessary

```
void BTree::insert(int key)
{
  if(root!=NULL)
    insert(key, root);
  else
  {
    root=new BTNode;
    root->key_value=key;
    root->left=NULL;
    root->right=NULL;
  }
}

void BTree::insert(int key, BTNode* leaf)
{
  if(key< leaf->key_value)
    {
      if(leaf->left!=NULL)
         insert(key, leaf->left);
      else
         {
           leaf->left=new BTNode;
           leaf->left->key_value=key;
           leaf->left->left=NULL;    //Set the left child of the child node to null
           leaf->left->right=NULL;   //Set the right child of the child node to null
         }
    }
  else if(key>=leaf->key_value)
    {
      if(leaf->right!=NULL)
         insert(key, leaf->right);
      else
         {
           leaf->right=new BTNode;
      leaf->right->key_value=key;
      leaf->right->left=NULL;  //Set the left child of the child node to null
      leaf->right->right=NULL; //Set the right child of the child node to null
         }
    }
}

BTNode* BTree::search(int key, BTNode *leaf)
{
  if(leaf!=NULL)
  {
    if(key==leaf->key_value)
      return leaf;
    if(key<leaf->key_value)
      return search(key, leaf->left);
    else
      return search(key, leaf->right);
  }
  else return NULL;
}
```

```
BTNode* BTree::search(int key)
{
  return search(key,root);
}

void BTree::destroy(BTNode *leaf)
{
  if(leaf!=NULL)
  {
    destroy(leaf->left);
    destroy(leaf->right);
    delete leaf;
  }
}

void BTree::destroy()
{
  destroy(root);
}

void preorderPrint( BTNode* btnp )
{
  if (btnp != NULL){
    cout<<btnp->key_value<<" ";
    preorderPrint(btnp->left);
    preorderPrint(btnp->right);
  }
  else {
    //Nothing to do
  }
}

int main(){
  BTree* bt=new BTree; //NOTE: "new" always returns a pointer
  bt->insert(12);
  bt->insert(4);
  bt->insert(10);
  bt->insert(21);
  bt->insert(13);
  bt->insert(3);
  bt->insert(15);
  bt->insert(22);
  preorderPrint(bt->root);
  cout<<endl;
  bt->destroy();
}
```

The above program creates a tree like this:



(This is the "ascii-art" representation of the tree)