BST

TreeNode.h

```
class TreeNode {
  friend class BST;

  public:
    TreeNode();
    TreeNode(int i, TreeNode*R, TreeNode*L);
    int getItem() const;

  private:
    int item;
    TreeNode *lChild;
    TreeNode *rChild;
};

TreeNode::TreeNode() {
  lChild = rChild = NULL;
}

TreeNode::TreeNode(int i, TreeNode *L=0, TreeNode *R=0)
  : item(i), lChild(L), rChild(R) { }

int TreeNode::getItem() const {
  return item;
}
```

BST.h

```
#include <iostream>
#include "TreeNode.h"

#ifndef BST_H
#define BST_H
class BST {
  public:
    BST();
    bool insert(int i);
    bool remove(int i);
    bool search(int i);
    std::ostream& travInOrder(std::ostream &out) ;
    std::ostream& travPreOrder(std::ostream &out) ;
    std::ostream& travPostOrder(std::ostream &out) ;

  private:
    TreeNode* root;
    int findMin(TreeNode* toFind);
    std::ostream& preOrder(std::ostream &out, TreeNode* cur) ;
    std::ostream& inOrder(std::ostream &out, TreeNode* cur) ;
    std::ostream& postOrder(std::ostream &out, TreeNode* cur) ;
    bool searchNode(TreeNode* cur, int i);

};
#endif
```

BST.cpp

```
#include <iostream>
```

```
2  #include "BST.h"
3
4  /**Constructor**/
5  BST::BST() :
6    root(0)
7  {}
8  /**End constructor**/
9
10 //Inserts a node into the bst
11 bool BST::insert(int i) {
12   if(root==0) {
13     root = new TreeNode(i,0,0);
14     return true;
15   }
16   TreeNode* current = root;
17   TreeNode* previous = 0;
18   while(current!=0) {
19     if(i==current->getItem()) return false;
20     previous = current;
21     if(i>current->getItem()) current = current->rChild;
22     else current = current->lChild;
23   }
24   if(i>previous->getItem()) previous->rChild = new TreeNode(i,0,0);
25   else previous->lChild = new TreeNode(i,0,0);
26   return true;
27 } //End insert()
28
29 bool BST::remove(int i) {
30   if(!search(i)) return false;
31   TreeNode* previous = 0;
32   TreeNode* current = root;
33   while(current->getItem()!=i) {
34     previous = current;
35     current = (current->getItem()>i)?current->lChild : current->rChild;
36   }
37   //Case 1: leaf node
38   if(current->lChild==0 && current->rChild==0) {
39     if(previous==0) root = 0;
40     else if (i<previous->getItem())
41       previous->lChild=0;
42     else previous->rChild=0;
43     delete current;
44   }
45   //Case 2: one child
46   else if(current->lChild==0) {
47     if(previous==0) root = current->rChild;
48     else if (i<previous->getItem())
49       previous->lChild=current->rChild;
50     else previous->rChild=current->rChild;
51     delete current;
52   }
53   else if(current->rChild==0) {
54     if(previous==0) root = current->lChild;
55     else if (i<previous->getItem())
56       previous->lChild=current->lChild;
57     else previous->rChild=current->rChild;
58     delete current;
59   }
60   else {
61     int min = BST::findMin(current);
62     BST::remove(min);
63     current->item = min;
64   }
65   return true;
66 } //End remove()
67
68 //aux function for remove()
69 int BST::findMin(TreeNode* toFind) {
```

```
  TreeNode* current = toFind->rChild;
  while(current->lChild!=0) current = current->lChild;
  return current->getItem();
} //End findMin()

//Returns true if the object is in the tree
bool BST::search(int i) {
  return BST::searchNode(root, i);
} //End search()
bool BST::searchNode(TreeNode* cur, int i) {
  if(cur==0) return false;
  if(cur->getItem()==i) return true;
  if(cur->getItem()>i)
    return BST::searchNode(cur->lChild, i);
  else return BST::searchNode(cur->rChild, i);
}


std::ostream& BST::travInOrder(std::ostream &out) {
  return BST::inOrder(out, root) << std::endl;
}
std::ostream& BST::travPreOrder(std::ostream &out) {
  return BST::preOrder(out, root) << std::endl;
}
std::ostream& BST::travPostOrder(std::ostream &out) {
  return BST::postOrder(out, root) << std::endl;
}

std::ostream& BST::preOrder(std::ostream &out, TreeNode* cur) {
  if(cur==0) return out;
  out << " " << cur->getItem();
  BST::preOrder(out, cur->lChild);
  BST::preOrder(out, cur->rChild);
  return out;
}
std::ostream& BST::inOrder(std::ostream &out, TreeNode* cur) {
  if(cur==0) return out;
  BST::inOrder(out, cur->lChild);
  out << " " << cur->getItem();
  BST::inOrder(out, cur->rChild);
  return out;
}
std::ostream& BST::postOrder(std::ostream &out, TreeNode* cur) {
  if(cur==0) return out;
  BST::postOrder(out, cur->lChild);
  BST::postOrder(out, cur->rChild);
  out << " " << cur->getItem();
  return out;
}
```

main.cpp

```
#include <iostream>
#include <stdlib.h>
#include "BST.cpp"
using namespace std;

int main() {
  BST* tree = new BST();
  srand (time(NULL));
  cout << "Testing insert, and inOrder Traversal" << endl;
  for(int i=0; i<20; i++) {
    int j = rand()%100;
    if(!tree->insert(j))
      cout<<"Unsuccessful addition: duplicate of " << j << endl;
```

```
  }

  cout << "testing inOrder transverse" << endl;
  tree->travInOrder(cout);
  cout << "testing preOrder traverse" << endl;
  tree->travPreOrder(cout);
  cout << "testing postOrder transverse" << endl;
  tree->travPostOrder(cout);

  cout << "Searching..." << "Found:" << endl;
  for(int i=0; i<=100; i++)
    if(tree->search(i)) cout << " "<< i;
  cout << endl;
  cout << "Deleting... Deleted: " <<endl;
  for(int i=0; i<=100; i++)
    if(tree->remove(i)) cout << " " << i;
  cout << endl;

  return 1;
}
```

Output

```
Testing insert, and inOrder Traversal
Unsuccessful addition: duplicate of 89
testing inOrder transverse
 1 7 14 24 25 29 33 37 38 44 52 62 68 76 85 87 89 94 95
testing preOrder traverse
 52 29 14 7 1 24 25 38 37 33 44 94 85 76 68 62 87 89 95
testing postOrder transverse
 1 7 25 24 14 33 37 44 38 29 62 68 76 89 87 85 95 94 52
Searching...Found:
 1 7 14 24 25 29 33 37 38 44 52 62 68 76 85 87 89 94 95
Deleting... Deleted:
 1 7 14 24 25 29 33 37 38 44 52 62 68 76 85 87 89 94 95
```