

```
1  /*
2  280
3  BST_Development
4  Ben Lohman, Elliot Shaw
5  */
6
7  #include <iostream>
8  #include <string>
9  #include <ctime>
10 using namespace std;
11
12 struct Node {
13     int data;
14     Node* left, * right;
15 };
16
17 class BST {
18 private:
19     Node* root;
20     Node* insert(int, Node*); //helper
21     int getSize(Node*); //helper
22     void displayInOrder(Node*); //helper
23     Node* find(int, Node*); //helper
24     int getHeight(Node*, int); //helper
25     void displayPreOrder(Node*); //helper
26     bool isLeaf(Node*);
27     int countLeaves(Node*); //helper
28 public:
29     void displayPreOrder();
30     int getHeight();
31     BST();
32     void insert(int);
33     void displayInOrder();
34     int getSize();
35     void load(int, int, int);
36     void insertNonRecursive(int);
37     Node* find(int);
38     int count(int);
39     void del(int);
40     int countLeaves();
41     int maxValue();
42 }; //BST class
43
44 //helper functions
45
46 Node* BST::insert(int v, Node* r) {
47     if (r == nullptr) {
48         r = new Node;
49         r->left = r->right = nullptr;
```

```
50     r->data = v;
51     return r;
52 }
53 else if (v < r->data) {
54     r->left = insert(v, r->left);
55     return r;
56 }
57 else {
58     r->right = insert(v, r->right);
59     return r;
60 }
61 } //insert helper
62
63 Node* BST::find(int v, Node* r) {
64     while (r != nullptr) {
65         if (r->data == v) {
66             return r;
67         }
68         else if (v < r->data) {
69             r = r->left;
70         }
71         else {
72             r = r->right;
73         }
74     }
75     return nullptr;
76 } //find helper
77
78 void BST::displayPreOrder(Node* r) {
79     if (r != nullptr)
80     {
81         cout << r->data << endl;
82         displayPreOrder(r->left);
83         displayPreOrder(r->right);
84     }
85 } //displayPreOrder helper
86
87 void BST::displayInOrder(Node* r) {
88     if (r != nullptr)
89     {
90         displayInOrder(r->left);
91         cout << r->data << endl;
92         displayInOrder(r->right);
93     }
94 } //displayInOrder helper
95
96 int BST::getHeight(Node* r, int c) {
97     if (r != nullptr)
98     {
```

```
99         c++;
100         int left = getHeight(r->left, c);
101         int right = getHeight(r->right, c);
102         if (left > right) {
103             c = left;
104         }
105         else {
106             c = right;
107         }
108     }
109     return c;
110 } //getHeight helper
111
112 bool BST::isLeaf(Node* r) {
113     if (r->right == nullptr && r->left == nullptr) {
114         return true;
115     }
116     return false;
117 } //isLeaf helper
118
119 int BST::countLeaves(Node* r) {
120     int c = 0;
121     if (isLeaf(r)) {
122         return 1;
123     }
124     if (r->left != nullptr) {
125         c += countLeaves(r->left);
126     }
127     if (r->right != nullptr) {
128         c += countLeaves(r->right);
129     }
130     return c;
131 } //countLeaves helper
132
133 int BST::getSize(Node* r) {
134     if (r == nullptr)
135         return 0;
136     else
137         return 1 + getSize(r->left) + getSize(r->right);
138 } //getSize helper
139
140 //constructors
141
142 BST::BST() {
143     root = nullptr;
144 } //BST
145
146
147 //setters
```

```
148
149 void BST::insert(int v) {
150     root = insert(v, root);
151 } //insert
152
153 void BST::load(int c, int min, int max) {
154     srand(time(NULL));
155     for (int i = 0; i < c; i++){
156         root = insert((rand()%(max-min)) +min, root);
157     }
158 }//load
159
160 void BST::insertNonRecursive(int v) {
161     Node* check = root;
162     Node* checkptr = nullptr;
163     while (check != nullptr) {
164         checkptr = check;
165         if (v < checkptr->data) {
166             check = check->left;
167         }
168         else {
169             check = check->right;
170         }
171     }
172     if(checkptr == nullptr){
173         root = new Node;
174         root->left = root->right = nullptr;
175         root->data = v;
176     }
177     else if (v < checkptr->data) {
178         checkptr->left = new Node;
179         checkptr->left->data = v;
180         checkptr->left->left = nullptr;
181         checkptr->left->right = nullptr;
182     }
183     else {
184         checkptr->right = new Node;
185         checkptr->right->data = v;
186         checkptr->right->left = nullptr;
187         checkptr->right->right = nullptr;
188     }
189 }//insertNonRecursive
190
191
192 //getters
193
194 int BST::getHeight() {
195     return getHeight(root, 0);
196 } //getHeight;
```

```
197
198 int BST::getSize() {
199     return getSize(root);
200 } //getSize
201
202 int BST::maxValue() {
203     //pre-req: the tree is not an empty tree
204     Node* r = root;
205     while (r->right != nullptr) {
206         r = r->right;
207     }
208     return r->data;
209 } //maxValue
210
211 //utility
212
213 Node* BST::find(int v) {
214     return find(v, root);
215 } //find
216
217 int BST::count(int v) {
218     Node* r = root;
219     int count = 0;
220     while (r != nullptr) {
221         if (r->data == v) {
222             count++;
223         }
224         if (v < r->data) {
225             r = r->left;
226         }
227         else {
228             r = r->right;
229         }
230     }
231     return count;
232 } //count
233
234 void BST::displayInOrder() {
235     displayInOrder(root);
236 } //displayInOrder
237
238 void BST::displayPreOrder() {
239     displayPreOrder(root);
240 } //displayPreOrder
241
242 int BST::countLeaves() {
243     return countLeaves(root);
244 } //countLeaves
245
```

```
246 void BST::del(int v) {
247     Node* t = find(v, root);
248     int tval = t->data;
249     Node* p= root;
250     if (t != root) {
251         while (p->left != t && p->right != t) {
252             if (tval < p->data) {
253                 p = p->left;
254             }
255             else {
256                 p = p->right;
257             }
258         }
259         if (p->left == t) {
260             p->left = nullptr;
261         }
262         else {
263             p->right = nullptr;
264         }
265     }
266     else {
267         root = nullptr;
268     }
269     Node* ip = nullptr;
270     while(!isLeaf(t)){
271         Node* i = t;
272
273         while (!isLeaf(i)) {
274             ip = i;
275             if (i->right != nullptr) {
276                 i = i->right;
277             }
278             else {
279                 i = i->left;
280             }
281         }
282         insert(i->data);
283         if (ip->right != nullptr) {
284             ip->right = nullptr;
285         }
286         else {
287             ip->left = nullptr;
288         }
289         free(i);
290     }
291     free(t);
292 }//delete
293
294 int main() {
```

```
295     BST bst1 = BST();
296     cout << "Size: " << bst1.getSize() << endl << endl;
297
298     bst1.insert(20);
299     bst1.displayInOrder();
300     cout << "Size: " << bst1.getSize() << endl << endl;
301
302     bst1.insert(10);
303     bst1.insert(30);
304     bst1.displayInOrder();
305     cout << "Size: " << bst1.getSize() << endl << endl;
306
307     bst1.insert(5);
308     bst1.insert(40);
309     bst1.insert(25);
310     bst1.displayInOrder();
311     cout << "Size: " << bst1.getSize() << endl << endl;
312
313     bst1.insert(0);
314     bst1.insert(2);
315     bst1.insert(-5);
316     bst1.insert(-2);
317     bst1.displayInOrder();
318     cout << "Size: " << bst1.getSize() << endl << endl;
319
320     BST bst2 = BST();
321     bst2.load(10, -20, 20);
322     bst2.displayInOrder();
323     cout << "Size: " << bst2.getSize() << endl << endl;
324     bst2.insertNonRecursive(20);
325     bst2.displayInOrder();
326     cout << "Size: " << bst2.getSize() << endl << endl;
327     cout << "address of 20: " << bst2.find(20) << endl << endl;
328     bst2.insert(45);
329     bst2.insert(45);
330     bst2.insert(45);
331     bst2.insert(45);
332     bst2.insert(45);
333     cout << "count of 45s: " << bst2.count(45) << endl << endl;
334     bst2.displayInOrder();
335     cout << endl;
336     bst2.del(20);
337     bst2.displayInOrder();
338     cout << endl;
339     cout << "height of bst2: " << bst2.getHeight() << endl << endl;
340     bst2.displayPreOrder();
341     cout << endl << "number of leaves: " << bst2.countLeaves() << endl << endl;
342     cout << "Max value: " << bst2.maxValue();
343 } //main
```