```cpp
/*
280
BST_Development_2
Elliot Shaw
*/

#include <iostream>
#include <string>
#include <ctime>
using namespace std;

struct Node {
    int data;
    Node* left, * right;
};

class BST {
private:
    Node* root;
    Node* insert(int, Node*); //helper
    int getSize(Node*); //helper
    void displayInOrder(Node*); //helper
    Node* find(int, Node*);//helper
    int getHeight(Node*, int);//helper
    void displayPreOrder(Node*);//helper
    bool isLeaf(Node*);
    int countLeaves(Node*);//helper
    void treeClear(Node*);//helper
    int sumInRange(int, int, Node*);//helper
public:
    BST();
    //setters
    void treeClear();
    void load(int, int, int);
    void insertNonRecursive(int);
    void del(int);
    //getters
    int getHeight();
    int getSize();
    int treeSize();
    int count(int);
    //utility
    int sumInRange(int, int);
    void displayPreOrder();
    void insert(int);
    void displayInOrder();
    Node* find(int);
    int countLeaves();
    int maxValue();
```

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

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

```
148
149 void BST::treeClear(Node* r) {
150     Node* runner = r;
151     if (runner->right != nullptr) {
152         treeClear(runner->right);
153         runner->right = nullptr;
154     }
155     if (runner->left != nullptr) {
156         treeClear(runner->left);
157         runner->left = nullptr;
158     }
159     free(runner);
160     //cout << "freed" << endl;
161 } //treeClear Helper
162
163 int BST::sumInRange(int min, int max, Node* r) {
164     int sum = 0;
165     if (r != nullptr) {
166         if (r->data <= max && r->data >= min) {
167             return (sum + r->data + sumInRange(min, max, r->left) + sumInRange
                  (min, max, r->right));
168         }
169         else {
170             return sum + sumInRange(min, max, r->left) + sumInRange(min, max,
                  r->right);
171         }
172     }
173     return 0;
174 }
175
176
177 //constructors
178
179 BST::BST() {
180     root = nullptr;
181 } //BST
182
183
184 //setters
185
186 void BST::insert(int v) {
187     root = insert(v, root);
188 } //insert
189
190 void BST::load(int c, int min, int max) {
191     srand(time(NULL));
192     for (int i = 0; i < c; i++) {
193         root = insert((rand() % (max - min + 1)) + min, root);
194     }
```

```
195 }//load
196
197 void BST::insertNonRecursive(int v) {
198     Node* check = root;
199     Node* checkptr = nullptr;
200     while (check != nullptr) {
201         checkptr = check;
202         if (v < checkptr->data) {
203             check = check->left;
204         }
205         else {
206             check = check->right;
207         }
208     }
209     if (checkptr == nullptr) {
210         root = new Node;
211         root->left = root->right = nullptr;
212         root->data = v;
213     }
214     else if (v < checkptr->data) {
215         checkptr->left = new Node;
216         checkptr->left->data = v;
217         checkptr->left->left = nullptr;
218         checkptr->left->right = nullptr;
219     }
220     else {
221         checkptr->right = new Node;
222         checkptr->right->data = v;
223         checkptr->right->left = nullptr;
224         checkptr->right->right = nullptr;
225     }
226 }//insertNonRecursive
227
228
229 //getters
230
231 int BST::getHeight() {
232     return getHeight(root, 0);
233 } //getHeight;
234
235 int BST::getSize() {
236     return getSize(root);
237 } //getSize
238
239 int BST::maxValue() {
240     //pre-req: the tree is not an empty tree
241     Node* r = root;
242     while (r->right != nullptr) {
243         r = r->right;
```

```
244     }
245     return r->data;
246 }//maxValue
247 
248 int BST::treeSize() {
249     return getSize(root);
250 }
251 
252 void BST::treeClear() {
253     if (root != nullptr) {
254         treeClear(root);
255     }
256     root = nullptr;
257 }//treeClear
258 
259 
260 //utility
261 
262 Node* BST::find(int v) {
263     return find(v, root);
264 }//find
265 
266 int BST::count(int v) {
267     Node* r = root;
268     int count = 0;
269     while (r != nullptr) {
270         if (r->data == v) {
271             count++;
272         }
273         if (v < r->data) {
274             r = r->left;
275         }
276         else {
277             r = r->right;
278         }
279     }
280     return count;
281 }//count
282 
283 void BST::displayInOrder() {
284     displayInOrder(root);
285 } //displayInOrder
286 
287 void BST::displayPreOrder() {
288     displayPreOrder(root);
289 }//displayPreOrder
290 
291 int BST::countLeaves() {
292     return countLeaves(root);
```

```
293 }//countLeaves
294
295 void BST::del(int v) {
296     Node* t = find(v, root);
297     int tval = t->data;
298     Node* p = root;
299     if (t != root) {
300         while (p->left != t && p->right != t) {
301             if (tval < p->data) {
302                 p = p->left;
303             }
304             else {
305                 p = p->right;
306             }
307         }
308         if (p->left == t) {
309             p->left = nullptr;
310         }
311         else {
312             p->right = nullptr;
313         }
314     }
315     else {
316         root = nullptr;
317     }
318     Node* ip = nullptr;
319     while (!isLeaf(t)) {
320         Node* i = t;
321
322         while (!isLeaf(i)) {
323             ip = i;
324             if (i->right != nullptr) {
325                 i = i->right;
326             }
327             else {
328                 i = i->left;
329             }
330         }
331         insert(i->data);
332         if (ip->right != nullptr) {
333             ip->right = nullptr;
334         }
335         else {
336             ip->left = nullptr;
337         }
338         free(i);
339     }
340     free(t);
341 }//delete
```

```
342
343 int BST::sumInRange(int min, int max) {
344     int sum = 0;
345     if (root != nullptr) {
346         return sumInRange(min, max, root);
347     }
348     else {
349         return sum;
350     }
351 }//sumInRange
352
353 int main() {
354     BST bst1 = BST();
355
356     int x[] = { 40, 10, 80, 70, 50, 30, 10, 90, 10, 60, 5, 25, 35 };
357     int upto = size(x);
358
359     for (int i = 0; i < upto; i++) {
360         bst1.insert(x[i]);
361     }
362
363     cout << "Tree before treeClear: \n"; bst1.displayInOrder();
364     cout << "Tree size: " << bst1.treeSize() << endl << endl;
365
366     cout << "sum in range [4,31]: " << bst1.sumInRange(4, 31) << endl;
367     cout << "should be 90 ..." << endl << endl;
368     cout << "sum in range [60,80]: " << bst1.sumInRange(60, 80) << endl;
369     cout << "should be 210 ..." << endl << endl;
370
371     bst1.treeClear();
372     cout << "Cleared the tree" << endl;
373     cout << "Tree size: " << bst1.treeSize() << endl;
374
375     /*
376     BST bst1 = BST();
377     cout << "Size: " << bst1.getSize() << endl << endl;
378
379     bst1.insert(20);
380     bst1.displayInOrder();
381     cout << "Size: " << bst1.getSize() << endl << endl;
382
383     bst1.insert(10);
384     bst1.insert(30);
385     bst1.displayInOrder();
386     cout << "Size: " << bst1.getSize() << endl << endl;
387
388     bst1.insert(5);
389     bst1.insert(40);
390     bst1.insert(25);
```

```
391     bst1.displayInOrder();
392     cout << "Size: " << bst1.getSize() << endl << endl;
393
394     bst1.insert(0);
395     bst1.insert(2);
396     bst1.insert(-5);
397     bst1.insert(-2);
398     bst1.displayInOrder();
399     cout << "Size: " << bst1.getSize() << endl << endl;
400
401     BST bst2 = BST();
402     bst2.load(10, -20, 20);
403     bst2.displayInOrder();
404     cout << "Size: " << bst2.getSize() << endl << endl;
405     bst2.insertNonRecursive(20);
406     bst2.displayInOrder();
407     cout << "Size: " << bst2.getSize() << endl << endl;
408     cout << "address of 20: " << bst2.find(20) << endl << endl;
409     bst2.insert(45);
410     bst2.insert(45);
411     bst2.insert(45);
412     bst2.insert(45);
413     bst2.insert(45);
414     cout << "count of 45s: " << bst2.count(45) << endl << endl;
415     bst2.displayInOrder();
416     cout << endl;
417     bst2.del(20);
418     bst2.displayInOrder();
419     cout << endl;
420     cout << "height of bst2: " << bst2.getHeight() << endl << endl;
421     bst2.displayPreOrder();
422     cout << endl << "number of leaves: " << bst2.countLeaves() << endl << endl;
423     cout << "Max value: " << bst2.maxValue(); */
424 } //main
425
```