

```

1  /*
2  Filename: p4.cpp
3  Author(s): Zachary Rea and Parker Ross
4  Date: 12 February 2023
5  Description: The cpp for Binary Search Tree
6  */
7  #include <iostream> //allows for usage of cin, cout, and cerr
8
9  #include "p4.h"
10
11 using namespace std;
12
13 //*****
14 //Constructors and De-constructors
15 //*****
16 //Constructor
17 //Written by Zach
18
19 sNode::sNode(string text) {
20     this->text = text;
21     left = right = NULL;
22     this->h = 0;
23 }
24
25 //*****
26 //Constructor
27 //Written by Zach
28
29 sBST::sBST() {
30     treeCount = 0;
31     root = NULL;
32 }
33
34 //*****
35 //De-constructor
36 //Written by Zach
37
38 sBST::~sBST() {
39     clear();
40 }
41
42 //*****
43 //Private Functions
44 //*****
45 //Function to find the minimum value of the subtree
46 //Written by Zach
47
48 string sBST::findMin(sNode *ptr) {
49     string rc;
50     if (ptr->left) {
51         findMin(ptr->left);
52     } else if (ptr->right) {
53         findMin(ptr->right);
54     } else {
55         rc = ptr->text;
56     }
57     return rc;
58 }
59
60 //*****
61 //Function for help with recursion
62 //Written by Zach
63
64 bool sBST::insert(sNode *p, string text) {
65     bool rc = false;
66     //Empty tree base case
67     if (treeCount == 0) {
68         sNode *t = new sNode(text);
69         root = t;

```

```

70         treeCount++;
71         rc = true;
72     } else {
73         //Compare the inserting string with the string of the node
74         int check = text.compare(p->text);
75         //Go to the right child
76         if (check > 0) {
77             if (p->right) {
78                 rc = insert(p->right, text);
79             } else {
80                 sNode *t = new sNode(text);
81                 p->right = t;
82                 treeCount++;
83                 rc = true;
84             }
85         } else if (check < 0) {
86             //Go to the left child
87             if (p->left) {
88                 rc = insert(p->left, text);
89             } else {
90                 sNode *t = new sNode(text);
91                 p->left = t;
92                 treeCount++;
93                 rc = true;
94             }
95         }
96     }
97     return rc;
98 }
99
100 //*****
101 //Function for help with recursion
102 //Written by Zach but stubbed for p4a
103
104 bool sBST::remove(sNode *p, string text) {
105     bool rc = false;
106     /*
107     if (p) {
108         if (text == p->text) {
109             if (p->right) {
110                 p->text = findMin(p->right);
111                 rc = remove(p->right, p->text);
112             } else if (p->left) {
113                 p->text = p->left->text;
114                 p->right = p->left->right;
115                 p->left = p->left->left;
116                 delete p->left;
117                 treeCount--;
118             } else {
119                 delete p;
120                 treeCount--;
121             }
122             rc = true;
123         } else if ((text < p->text) && (p->left)) {
124             rc = remove(p->left, text);
125         } else if ((text > p->text) && (p->right)) {
126             rc = remove(p->right, text);
127         }
128     }
129     */
130     return rc;
131 }
132
133 //*****
134 //Function for help with recursion
135 //Written by Parker
136
137 bool sBST::isIn() {
138     return false;

```

```

139     }
140
141     //*****
142     //Function for help with recursion
143     //Written by Parker
144
145     void sBST::printIt(sNode *p) const{
146         if (p != NULL){ //making sure function has a printable node
147             // prints left subtree, root then right subtree for ascending order
148             printIt(p->left); //prints Left subtree
149             cout << p->text << endl; //print root
150             printIt(p->right); //prints right subtree
151         }
152     }
153
154     //*****
155     //Function for help with recursion
156     //Written by Zach
157
158     void sBST::clear(sNode *p) {
159         if (p) {
160             clear(p->left);
161             clear(p->right);
162             delete p;
163         }
164     }
165
166     //*****
167     //Public Functions
168     //*****
169     //Function to insert the text into the tree
170     //Written by Zach
171
172     bool sBST::insert(string text) {
173         bool rc = insert(root, text);
174         return rc;
175     }
176
177     //*****
178     //Function to remove the node with the given text
179     //Written by Zach
180
181     bool sBST::remove(string text) {
182         bool rc = false;
183         if (treeCount) {
184             rc = remove(root, text);
185         }
186         return rc;
187     }
188
189     //*****
190     //Function to tell if the tree contains the given text
191     //Written by Parker
192
193     bool sBST::isIn(string text) {
194         return false;
195     }
196
197     //*****
198     //Function to print the BST values in ascending order
199     //Written by Parker
200
201     void sBST::printIt() const{
202         if (treeCount) {
203             printIt(root);
204         }
205     }
206
207     //*****

```

```
208 //Function to show the number of nodes in the tree
209 //Written by Zach
210
211 int sBST::count() const{
212     return treeCount;
213 }
214
215 //*****
216 //Function to remove all of the nodes in the tree
217 //Written by Zach
218
219 void sBST::clear() {
220     clear(root);
221     treeCount = 0;
222     root = NULL;
223 }
```