```cpp
/*
Filename: p5.cpp
Author(s): Zachary Rea and Parker Ross
Date: 19 February 2023
Description: The cpp for Binary Search Tree AVL
*/
#include <iostream>  //allows for usage of cin, cout, and cerr

#include "p5.h"

using namespace std;

//*************************************************************************
//Constructors and De-constructors
//*************************************************************************
//Constructor
//Written by Zach edited by Parker

sNode::sNode(string text) {
    this->text = text;
    left = right = NULL;
    h = 1;
}

//*************************************************************************
//Constructor
//Written by Zach

sAVL::sAVL() {
    treeCount = 0;
    root = NULL;
}

//*************************************************************************
//De-constructor
//Written by Zach

sAVL::~sAVL() {
    clear(root);
}

//*************************************************************************
//Private Functions
//*************************************************************************
//Function to find the minimum value of the subtree
//Written by Zach edited by Parker

string sAVL::findMin(sNode *ptr) const {
    string rc;
    if (ptr->left) {
        //look for left child
        rc = findMin(ptr->left);
    } else {
        rc = ptr->text;
    }
    return rc;
}

//*************************************************************************
//Function for help with recursion
//Written by Zach

bool sAVL::insert(sNode *&p, string text) {
    bool rc = false;
    if (p) {
        if (text > p->text) {
            rc = insert(p->right, text);
        } else if (text < p->text) {
            rc = insert(p->left, text);
```

```
70          }
71      } else {
72          p = new sNode (text);
73          treeCount++;
74          rc = true;
75      }
76      bal(p);
77      return rc;
78  }
79
80  //*************************************************************************
81  //Function for help with recursion
82  //Written by Zach edited by Parker
83
84  bool sAVL::remove(sNode *&p, string text) {
85      bool rc = false;
86      if (p) {
87          //case for the correct node
88          if (text == p->text) {
89              //replaces min of right child
90              if (p->right) {
91                  p->text = findMin(p->right);
92                  rc = remove(p->right,p->text);
93                  //simply replace with left child if no right child
94              } else if (p->left) {
95                  sNode *t = p;
96                  p = p->left;
97                  delete t;
98                  treeCount--;
99                  rc = true;
100             } else {
101                 delete p;
102                 treeCount--;
103                 rc =true;
104                 p = NULL;
105             }
106             //recursion for children
107         } else if (text < p->text) {
108             rc = remove(p->left,text);
109         } else if (text > p->text) {
110             rc = remove(p->right,text);
111         }
112         bal(p);
113     }
114     return rc;
115 }
116
117 //*************************************************************************
118 //Function for help with recursion
119 //Written by Parker
120 bool sAVL::isIn(sNode *p, string text) const {
121     bool rc = false;
122     if (p){
123         if (text < p->text) {
124             rc = isIn(p->left,text);
125         } else if (text > p->text) {
126             rc = isIn (p-> right,text);
127         } else {
128         rc = true;
129         }
130     }
131     return rc;
132 }
133
134 //*************************************************************************
135 //Function for help with recursion
136 //Written by Parker
137
138 void sAVL::printIt(sNode *p, int &index) const{
```

```
139       if (p){
140       // prints left subtree, root then right subtree for ascending order
141           printIt(p->left,index);
142           cout << "At " << index << " the string is " <<p->text << ": height = "
143           << p->h << endl;
144           index++;
145           printIt(p->right,index);
146       }
147   }
148
149   //*****************************************************************************
150   //Function for help with recursion
151   //Written by Zach
152
153   void sAVL::clear(sNode *p) {
154       if (p) {
155           clear(p->left);
156           clear(p->right);
157           delete p;
158       }
159   }
160
161   //*****************************************************************************
162   //Function to rotate node to the left
163   //Written by Zach
164
165   void sAVL::rotateLeft(sNode *&p1) {
166       sNode *p2 = p1->right;
167       p1->right = p2->left;
168       p2->left = p1;
169       p1->h = calcHeight(p1);
170       p2->h = calcHeight(p2);
171       p1 = p2;
172   }
173
174   //*****************************************************************************
175   //Function to rotate the node to the right
176   //Written by Zach
177
178   void sAVL::rotateRight(sNode *&p1) {
179
180       sNode *p2 = p1->left;
181       p1->left = p2->right;
182       p2->right = p1;
183       p1->h = calcHeight(p1);
184       p2->h = calcHeight(p2);
185       p1 = p2;
186   }
187
188   //*****************************************************************************
189   ////Function to balance a node
190   //Written by Zach
191
192   void sAVL::bal(sNode *&p) {
193
194       if (p) {
195           int diff = height(p->left) - height(p->right);
196           if (diff == 2) {
197               diff = height(p->left->left) - height(p->left->right);
198               if (diff < 0) {
199                   rotateLeft(p->left);
200               }
201               rotateRight(p);
202           } else if (diff == -2) {
203               if (height(p->right->left) > height(p->right->right)) {
204                   rotateRight(p->right);
205               }
206               rotateLeft(p);
207           }
```

```
208             p->h = calcHeight(p);
209         }
210     }
211
212     //*************************************************************************
213     //Function to return the height of a node
214     //Written by Zach
215
216     int sAVL::height(sNode *p) const{
217         int rc = 0;
218         if (p) {
219             rc = p->h;
220         }
221         return rc;
222     }
223
224     //*************************************************************************
225     //Function to calculate the height of a node based on children
226     //Written by Zach
227
228     int sAVL::calcHeight(sNode *p)const {
229         return (max(height(p->left),height(p->right)) + 1);
230     }
231
232     //*************************************************************************
233     //Public Functions
234     //*************************************************************************
235     //Function to insert the text into the tree
236     //Written by Zach
237
238     bool sAVL::insert(string text) {
239         return (insert(root, text));
240     }
241
242     //*************************************************************************
243     //Function to remove the node with the given text
244     //Written by Zach
245
246     bool sAVL::remove(string text) {
247         return (remove(root, text));
248     }
249
250     //*************************************************************************
251     //Function to tell if the tree co`ntains the given text
252     //Written by Parker
253
254     bool sAVL::isIn(string text) const {
255         return (isIn(root, text));
256     }
257
258     //*************************************************************************
259     //Function to print the BST values in ascending order
260     //Written by Parker
261
262     void sAVL::printIt() const{
263         int index = 0;
264         printIt(root,index);
265     }
266
267     //*************************************************************************
268     //Function to show the number of nodes in the tree
269     //Written by Zach
270
271     int sAVL::count() const{
272         return treeCount;
273     }
274
275     //*************************************************************************
276     //Function to remove all of the nodes in the tree
```

```
277  //Written by Zach
278
279  void sAVL::clear() {
280      clear(root);
281      treeCount = 0;
282      root = NULL;
283  }
284
285  //*************************************************************************
286  //Non-member functions
287  //*************************************************************************
288  //Function to determine the max value of two integers
289  //Written by Zach
290
291  int max(int a, int b) {
292      int rc;
293      if (a > b) {
294          rc = a;
295      } else if (a < b) {
296          rc = b;
297      } else {
298          rc = 0;
299      }
300      return rc;
301  }
```