```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);
        }
    } else {
        p = new sNode (text);
        treeCount++;
        rc = true;
    }
    bal(p);
    return rc;
}

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

bool sAVL::remove(sNode *&p, string text) {
    bool rc = false;
    if (p) {
        //case for the correct node
        if (text == p->text) {
            //replaces min of right child
            if (p->right) {
                p->text = findMin(p->right);
                rc = remove(p->right,p->text);
                //simply replace with left child if no right child
            } else if (p->left) {
                sNode *t = p;
                p = p->left;
                delete t;
                treeCount--;
                rc = true;
            } else {
                delete p;
                treeCount--;
                rc =true;
                p = NULL;
            }
            //recursion for children
        } else if (text < p->text) {
            rc = remove(p->left,text);
        } else if (text > p->text) {
            rc = remove(p->right,text);
        }
        bal(p);
    }
    return rc;
}

//*******************************************************************************
//Function for help with recursion
//Written by Parker
bool sAVL::isIn(sNode *p, string text) const {
    bool rc = false;
    if (p){
        if (text < p->text) {
            rc = isIn(p->left,text);
        } else if (text > p->text) {
            rc = isIn (p-> right,text);
        } else {
        rc = true;
        }
    }
    return rc;
}

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

void sAVL::printIt(sNode *p, int &index) const{
```

```cpp
    if (p){
    // prints left subtree, root then right subtree for ascending order
        printIt(p->left,index);
        cout << "At " << index << " the string is " <<p->text << ": height = "
        << p->h << endl;
        index++;
        printIt(p->right,index);
    }
}

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

void sAVL::clear(sNode *p) {
    if (p) {
        clear(p->left);
        clear(p->right);
        delete p;
    }
}

//****************************************************************************
//Function to rotate node to the left
//Written by Zach

void sAVL::rotateLeft(sNode *&p1) {
    sNode *p2 = p1->right;
    p1->right = p2->left;
    p2->left = p1;
    p1->h = calcHeight(p1);
    p2->h = calcHeight(p2);
    p1 = p2;
}

//****************************************************************************
//Function to rotate the node to the right
//Written by Zach

void sAVL::rotateRight(sNode *&p1) {

    sNode *p2 = p1->left;
    p1->left = p2->right;
    p2->right = p1;
    p1->h = calcHeight(p1);
    p2->h = calcHeight(p2);
    p1 = p2;
}

//****************************************************************************
////Function to balance a node
//Written by Zach

void sAVL::bal(sNode *&p) {

    if (p) {
        int diff = height(p->left) - height(p->right);
        if (diff == 2) {
            diff = height(p->left->left) - height(p->left->right);
            if (diff < 0) {
                rotateLeft(p->left);
            }
            rotateRight(p);
        } else if (diff == -2) {
            if (height(p->right->left) > height(p->right->right)) {
                rotateRight(p->right);
            }
            rotateLeft(p);
        }
        p->h = calcHeight(p);
    }
}

//******************************************************************************
//Function to return the height of a node
//Written by Zach

int sAVL::height(sNode *p) const{
    int rc = 0;
    if (p) {
        rc = p->h;
    }
    return rc;
}

//******************************************************************************
//Function to calculate the height of a node based on children
//Written by Zach

int sAVL::calcHeight(sNode *p)const {
    return (max(height(p->left),height(p->right)) + 1);
}

//******************************************************************************
//Public Functions
//******************************************************************************
//Function to insert the text into the tree
//Written by Zach

bool sAVL::insert(string text) {
    return (insert(root, text));
}

//******************************************************************************
//Function to remove the node with the given text
//Written by Zach

bool sAVL::remove(string text) {
    return (remove(root, text));
}

//******************************************************************************
//Function to tell if the tree co`ntains the given text
//Written by Parker

bool sAVL::isIn(string text) const {
    return (isIn(root, text));
}

//******************************************************************************
//Function to print the BST values in ascending order
//Written by Parker

void sAVL::printIt() const{
    int index = 0;
    printIt(root,index);
}

//******************************************************************************
//Function to show the number of nodes in the tree
//Written by Zach

int sAVL::count() const{
    return treeCount;
}

//******************************************************************************
//Function to remove all of the nodes in the tree
```

```
//Written by Zach

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

//******************************************************************************
//Non-member functions
//******************************************************************************
//Function to determine the max value of two integers
//Written by Zach

int max(int a, int b) {
    int rc;
    if (a > b) {
        rc = a;
    } else if (a < b) {
        rc = b;
    } else {
        rc = 0;
    }
    return rc;
}
```