```cpp
/*
Filename: p4.cpp
Author(s): Zachary Rea and Parker Ross
Date: 12 February 2023
Description: The cpp for Binary Search Tree
*/
#include <iostream>  //allows for usage of cin, cout, and cerr

#include "p4.h"

using namespace std;

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

sNode::sNode(string text) {
    this->text = text;
    left = right = NULL;
    this->h = 0;
}

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

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

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

sBST::~sBST() {
    clear();
}

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

string sBST::findMin(sNode *ptr) {
    string rc;
    if (ptr->left) {
        findMin(ptr->left);
    } else if (ptr->right) {
        findMin(ptr->right);
    } else {
        rc = ptr->text;
    }
    return rc;
}

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

bool sBST::insert(sNode *p, string text) {
    bool rc = false;
    //Empty tree base case
    if (treeCount == 0) {
        sNode *t = new sNode(text);
        root = t;
        treeCount++;
        rc = true;
    } else {
        //Compare the inserting string with the string of the node
        int check = text.compare(p->text);
        //Go to the right child
        if (check > 0) {
            if (p->right) {
                rc = insert(p->right,text);
            } else {
                sNode *t = new sNode(text);
                p->right = t;
                treeCount++;
                rc = true;
            }
        } else if (check < 0) {
            //Go to the left child
            if (p->left) {
                rc = insert(p->left,text);
            } else {
                sNode *t = new sNode(text);
                p->left = t;
                treeCount++;
                rc = true;
            }
        }
    }
    return rc;
}

//****************************************************************************
//Function for help with recursion
//Written by Zach but stubbed for p4a

bool sBST::remove(sNode *p, string text) {
    bool rc = false;
    /*
    if (p) {
        if (text == p->text) {
            if (p->right) {
                p->text = findMin(p->right);
                rc = remove(p->right,p->text);
            } else if (p->left) {
                p->text = p->left->text;
                p->right = p->left->right;
                p->left = p->left->left;
                delete p->left;
                treeCount--;
            } else {
                delete p;
                treeCount--;
            }
            rc = true;
        } else if ((text < p->text) && (p->left)) {
            rc = remove(p->left,text);
        } else if ((text > p->text) && (p->right)) {
            rc = remove(p->right,text);
        }
    }
    */
    return rc;
}

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

bool sBST::isIn() {
    return false;
```

```cpp
}

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

void sBST::printIt(sNode *p) const{
  if (p != NULL){ //making sure function has a printable node
    // prints left subtree, root then right subtree for ascending order
    printIt(p->left); //prints Left subtree
    cout << p->text << endl; //print root
    printIt(p->right); //prints right subtree
    }
}

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

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

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

bool sBST::insert(string text) {
    bool rc = insert(root, text);
    return rc;
}

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

bool sBST::remove(string text) {
    bool rc = false;
    if (treeCount) {
        rc = remove(root,text);
    }
    return rc;
}

//*****************************************************************************
//Function to tell if the tree contains the given text
//Written by Parker

bool sBST::isIn(string text) {
    return false;
}

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

void sBST::printIt() const{
    if (treeCount) {
        printIt(root);
    }
}

//*****************************************************************************
```

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

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

//****************************************************************************
//Function to remove all of the nodes in the tree
//Written by Zach

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