```
1 #ifndef ASSIGNMENT_3_BST_H
 2 #define ASSIGNMENT_3_BST_H
 3
 4 #include "Node.h"
 5
 6 class BST {
       using NodePtr = Node *;
 7
 8
 9 private:
       NodePtr m_root{nullptr};
10
11
       NodePtr insert(std::string &, NodePtr &);
12
13
       void print_tree(std::ostream &, NodePtr &,
14
   int);
15
16 public:
       int height(NodePtr &);
17
18
       int max(int, int);
19
20
       int balance_factor(NodePtr &);
21
22
23
       NodePtr rotate_right(NodePtr &);
24
25
       NodePtr rotate_left(NodePtr &);
26
       void insert(std::string &);
27
28
       static int compare(std::string &, NodePtr &);
29
30
       bool find(std::string &);
31
32
33
       friend std::ostream &operator<<(std::ostream</pre>
    &, BST &);
34 };
35
36 #endif //ASSIGNMENT_3_BST_H
```

```
File - C:\Users\Owen Jones\Desktop\School_Files\Winter2023\PROG2400_Files\assignment-3-OwenJRJones\src\Node.h
 1 #ifndef ASSIGNMENT_3_NODE_H
 2 #define ASSIGNMENT_3_NODE_H
 3
 4 #include <iomanip>
 5
 6 struct Node {
        using NodePtr = Node *;
 8
         std::string m_data{"NULL"};
 9
         NodePtr m_left{nullptr};
10
         NodePtr m_right{nullptr};
11
12 };
13
14 #endif //ASSIGNMENT_3_NODE_H
15
```

```
1 #include "BST.h"
 2 #include <iostream>
 3
 4 using namespace std;
 5
 6 int BST::height(NodePtr &node) {
      // Return height of subtree
       if (node == nullptr)
 8
 9
           return -1;
       return max(height(node->m_left), height(node
10
   ->m_right)) + 1;
11 }
12
13 int BST::max(int a, int b) {
      // Return greater value of two ints
14
      return (a > b) ? a : b;
15
16 }
17
18 int BST::balance_factor(NodePtr &node) {
       // Return balance factor of left and right
19
   subtrees of given node
       if (node == nullptr)
20
21
           return 0;
22
       return height(node->m_left) - height(node->
  m_right);
23 }
24
25 Node::NodePtr BST::rotate_right(NodePtr &y) {
26
       // Rotate left
27
       NodePtr x = y->m_left;
       NodePtr T2 = x->m_right;
28
29
       // Rotate values
       x->m_right = y;
30
31
       y-m_left = T2;
32
33
       return x;
34 }
35
```

```
36 Node::NodePtr BST::rotate_left(NodePtr &x) {
37
       // Rotate right
       NodePtr y = x->m_right;
38
39
       NodePtr T2 = y->m_left;
40
       // Rotate values
41
       y->m_left = x;
       x->m_right = T2;
42
43
44
       return y;
45 }
46
47 void BST::insert(string &word) {
48
       // Use recursive insert function/update root
   node
49
       m_root = insert(word, m_root);
50 }
51
52 Node::NodePtr BST::insert(string &word, NodePtr &
   node) {
       // Recursive insert function
53
       if (node == nullptr) {
54
           // Create new node and store data
55
56
           node = new Node();
           node->m_data = word;
57
58
59
           // Return new node
60
           return node;
61
       } else if (word.compare(node->m_data) < 0) {</pre>
62
           // Go left
63
           node->m_left = insert(word, node->m_left
64
   );
       } else if (word.compare(node->m_data) > 0) {
65
66
           // Go right
           node->m_right = insert(word, node->
67
   m_right);
       } else {
68
69
           return node;
```

```
70
        }
 71
 72
        // Run balancing
        int bf = balance_factor(node);
 73
 74
 75
        // Left-Left rotate
        if (bf > 1 && compare(word, node->m_left) <</pre>
 76
    0) {
 77
            return rotate_right(node);
 78
        }
 79
        //Right-Right rotate
 80
        if (bf < -1 && compare(word, node->m_right
    ) > 0) {
 81
            return rotate_left(node);
 82
        }
 83
        // Left-Right rotate
        if (bf > 1 && compare(word, node->m_left) >
 84
    0) {
 85
            node->m_left = rotate_left(node->m_left
    );
            return rotate_right(node);
 86
 87
        }
        // Right-Left rotate
 88
        if (bf < -1 && compare(word, node->m_right
 89
    ) < 0) {}
 90
            node->m_right = rotate_right(node->
    m_right);
 91
            return rotate_left(node);
        }
 92
 93
 94
        // Return original node
 95
        return node;
96 }
 97
 98 bool BST::find(string &word) {
        // Check if word is in dictionary
 99
100
        NodePtr node = m_root;
101
        bool found = false;
```

```
102
        // Look for desired node
103
        while (node != nullptr) {
104
            if (compare(word, node) < 0) {</pre>
105
                // Continue left
106
107
                node = node->m_left;
            } else if (compare(word, node) > 0) {
108
109
                // Continue right
110
                node = node->m_right;
            } else if (compare(word, node) == 0) {
111
112
                // Found desired node
113
                found = true;
114
                break;
            }
115
116
        }
117
        return found;
118 }
119
120 int BST::compare(string &word, NodePtr &node) {
        // Use string compare() to determine string
121
    comparison
        return word.compare(node->m_data);
122
123 }
124
125 void BST::print_tree(ostream &output, NodePtr &
    node, int indent) {
        // Recursive printing function
126
        if (node != nullptr) {
127
            // Pass in right side first
128
            print_tree(output, node->m_right, indent
129
     + 4);
130
            // Add current node to output
            output << setw(indent + node->m_data.
131
    length());
132
            output << node->m_data << endl;</pre>
            // Pass in left side
133
            print_tree(output, node->m_left, indent
134
     + 4);
```

```
File - C:\Users\Owen Jones\Desktop\School_Files\Winter2023\PROG2400_Files\assignment-3-OwenJRJones\src\BST.cpp
135
136 }
137
138 ostream &operator<<(ostream &output, BST &bst) {
          // Use recursive print_tree function
139
140
          bst.print_tree(output, bst.m_root, 0);
141
          return output;
142 }
143
```

```
1 #include <iostream>
 2 #include "SpellChecker.h"
 3
 4 using namespace std;
 5
 6 int main(int argc, char **argv) {
 7
       SpellChecker sc;
 8
 9
10
       string input_file;
11
       string output_file = "../output/output.txt";
       string dictionary = "../docs/dictionary.txt";
12
13
       switch (argc) {
14
15
            case 4:
16
                input_file = arqv[1];
                output_file = arqv[2];
17
                dictionary = argv[3];
18
19
                break;
20
            case 3:
21
                input_file = arqv[1];
                output_file = arqv[2];
22
23
                break;
24
            case 2:
25
                input_file = arqv[1];
26
                break;
27
            default:
                // Prompt user for file name(s) if
28
   not given
29
                cout << "Incorrect number / improper</pre>
   file types specified." << endl;</pre>
30
                cout << "Please specify relative path</pre>
31
    to input file: ";
32
                getline(cin, input_file);
33
34
                string input;
35
```

```
36
                cout << "Please specify relative path</pre>
    to output file or '-d'"
                     << " for default output file: ";
37
38
                getline(cin, input);
39
40
                if (input != "-d") {
41
                    output_file = input;
42
                }
43
                cout << "Please specify relative path</pre>
44
    to dictionary file or '-d'"
                     << " for default dictionary file
45
                getline(cin, input);
46
47
48
                if (input != "-d") {
                    dictionary = input;
49
50
                }
       }
51
52
53
       // Run SpellCheck
       sc.check_words(input_file, output_file,
54
   dictionary);
55
56
       return 0;
57 }
```

```
File - C:\Users\Owen Jones\Desktop\School Files\Winter2023\PROG2400 Files\assignment-3-OwenJRJones\src\SpellChecke
 1 #ifndef ASSIGNMENT_3_SPELLCHECKER_H
 2 #define ASSIGNMENT_3_SPELLCHECKER_H
 3
 4 #include "BST.h"
 5
 6 class SpellChecker {
 7 private:
        BST m_bst;
 8
 9 public:
        ~SpellChecker();
10
11
12
        void check_words(std::string &, std::string
     &, std::string &);
13
        void read_dictionary(std::string &);
14
15
        std::string remove_chars(std::string &);
16
17
18
        void save_tree(std::string &);
19
20
        void display_tree();
21 };
22
23 #endif //ASSIGNMENT_3_SPELLCHECKER_H
```

```
1 #include "SpellChecker.h"
 2 #include<iostream>
 3 #include <fstream>
 4
 5 using namespace std;
 6
 7 void SpellChecker::check_words(string &input_file
   , string &output_file,
                                    string &dict_file
 8
   ) {
 9
10
       // Read in dictionary words
11
       read_dictionary(dict_file);
12
       fstream file;
13
14
       string word;
15
16
       try {
           // Open text file
17
           file.open(input_file.c_str());
18
19
           if (file.is_open()) {
20
               // Loop through words
21
22
               while (file >> word) {
23
                    // Wash words of any special
   chars & convert to lowercase
24
                    word = remove_chars(word);
                    // Check if word is empty after
25
   wash
26
                    if (!word.empty()) {
                        // Check if word is found in
27
   BST, output if not
                        if (!m_bst.find(word)) {
28
29
                            cout << word << endl;</pre>
30
                        }
                    }
31
               }
32
33
           }
```

```
} catch (exception &e) {
34
           cout << e.what();</pre>
35
36
       }
37
       file.close();
38
       // Save BST to file
39
       save_tree(output_file);
40
41
42
       cout << endl << "BST saved to " <<
   output_file << "." << endl;
43 }
44
45 void SpellChecker::read_dictionary(string &
   file_name) {
46
       fstream file;
47
       string word;
48
49
       try {
           // Open dictionary file
50
           file.open(file_name.c_str());
51
52
           if (file.is_open()) {
               // Populate BST with dictionary words
53
               while (file >> word) {
54
                    m_bst.insert(word);
55
                }
56
57
       } catch (exception &e) {
58
           cout << e.what();</pre>
59
60
       file.close();
61
62 }
63
64 string SpellChecker::remove_chars(string &word) {
       // Loop through word and erase special chars
65
       for (auto i = 0; i < word.size(); i++) {</pre>
66
           if (word[i] < 'A' || word[i] > 'Z' &&
67
68
                                  word[i] < 'a' ||
   word[i] > 'z') {
```

```
69
                 // Erase non alphabetic char
                 word.erase(i, 1);
 70
 71
                 i--;
 72
             }
            // Convert chars to lowercase
 73
            word[i] = tolower(word[i]);
 74
 75
        }
 76
        return word;
 77 }
 78
 79 void SpellChecker::save_tree(string &file_name
    ) {
 80
        fstream file;
 81
 82
        try {
 83
            file.open(file_name.c_str(), ios::out |
    ios::trunc);
 84
            if (file.is_open()) {
 85
                 // Save BST to file
 86
 87
                 file << m_bst;
 88
 89
        } catch (exception &e) {
 90
             cout << e.what();</pre>
 91
 92
        file.close();
 93 }
 94
 95 void SpellChecker::display_tree() {
        // Display tree to console
 96
        cout << m_bst << endl;</pre>
 97
 98 }
 99
100 SpellChecker::~SpellChecker() = default;
```