```
1 #ifndef ASSIGNMENT_2_GRIDPOS_H
2 #define ASSIGNMENT_2_GRIDPOS_H
 3
 4 struct GridPos {
       int m_row{-1};
5
      int m_col{-1};
7 };
8
9 #endif //ASSIGNMENT_2_GRIDPOS_H
10
```

```
1 #include "MazeSolver.h"
 2 #include <iostream>
 3
 4 using std::string, std::cout, std::cin;
 6 int main(int argc, char **argv) {
       MazeSolver maze{};
 8
 9
       string file_name;
10
       // Prompt user for filename if one was not
11
   given, then open/read file
12
       if (arqc < 2) {
           cout << "No file path was given, please</pre>
13
   specify: ";
14
           getline(cin, file_name);
15
       } else {
           file_name = argv[1];
16
       }
17
18
19
       // Solve maze and display output to console
       maze.solve(file_name);
20
21
       maze.display_maze();
22
23
       return 0;
24 }
```

```
1 #include "MazeSolver.h"
 2 #include "Stack.h"
 3 #include <iostream>
 4 #include <fstream>
 5 #include <filesystem>
 6
 7 using namespace std;
 8
 9 MazeSolver::~MazeSolver() = default;
10
11 void MazeSolver::read_file(string &path) {
       fstream target_file(path);
12
       string line;
13
       auto count{0};
14
15
       // Open file and read in maze
16
17
       try {
           if (target_file.is_open()) {
18
               // Populate Maze 2D array with file
19
   data
20
               while (getline(target_file, line)) {
                    for (auto i = 0; i < line.length</pre>
21
   (); i++) {
22
                        m_maze[count][i] = line[i];
23
                    }
24
                    count++;
25
                }
26
       } catch (exception &e) {
27
           cout << e.what() << endl;</pre>
28
29
       }
30
       target_file.close();
31 }
32
33 void MazeSolver::save_solution(string &path) {
       // Save solution to file
34
35
       fstream target_file;
36
       filesystem::path p(path);
```

```
string file_name = "..\\solved\\" + p.stem().
37
   string() + "_solution.txt";
38
39
       try {
            target_file.open(file_name, ios::out |
40
   ios::trunc);
41
            if (target_file.is_open()) {
42
43
                // Output maze to file
                for (auto i = 0; i < MAZE_WIDTH; i</pre>
44
   ++) {
45
                     for (auto j = 0; j < MAZE_LENGTH</pre>
   ; j++) {
46
                         target_file << m_maze[i][j];</pre>
47
                     }
                     if (i != MAZE_LENGTH - 1) {
48
                         target_file << '\n';</pre>
49
                     }
50
51
            } else {
52
53
                cout << "Failed to save solution."</pre>
    << endl;
54
       } catch (exception &e) {
55
            cout << e.what();</pre>
56
       }
57
58
       target_file.close();
       cout << "Solution saved to: " << file_name</pre>
59
    << endl;
60 }
61
62 void MazeSolver::solve(string &path) {
63
       bool solved = false;
64
       Stack pos;
       GridPos curr_loc{1, 0};
65
66
       // Read file data into maze
67
68
       read_file(path);
```

```
69
70
       // Loop to find correct path through maze
       while (!solved) {
71
72
73
           // Check if maze was solved
74
           if (curr_loc.m_row == MAZE_LENGTH - 2 &&
               curr_loc.m_col == MAZE_WIDTH - 1) {
75
               pos.push(curr_loc);
76
77
               solved = true;
78
           }
79
80
           // Check to the right of current
   position (East)
           if (m_maze[curr_loc.m_row][curr_loc.
81
   m_col + 1] == ' ') {
               pos.push(curr_loc);
82
               m_maze[curr_loc.m_row][curr_loc.
83
   m_col] = '#';
84
               curr_loc.m_col++;
85
               continue;
86
           }
87
88
           // Check below the current position (
   South)
           if (m_maze[curr_loc.m_row + 1][curr_loc.
89
   m_col] == ' ') {
90
               pos.push(curr_loc);
               m_maze[curr_loc.m_row][curr_loc.
91
   m coll = '#';
92
               curr_loc.m_row++;
               continue;
93
94
           }
95
           // Check to the left of the current
96
   position (West)
           if (m_maze[curr_loc.m_row][curr_loc.
97
   m_col - 1] == ' ') {
               pos.push(curr_loc);
98
```

```
99
                m_maze[curr_loc.m_row][curr_loc.
    m_col] = '#';
100
                curr_loc.m_col--;
101
                continue;
            }
102
103
            // Check above the current position (
104
    North)
            if (m_maze[curr_loc.m_row - 1][curr_loc.
105
    m_col] == ' ') {
106
                pos.push(curr_loc);
                m_maze[curr_loc.m_row][curr_loc.
107
    m coll = '#';
108
                curr_loc.m_row--;
                continue;
109
            }
110
111
112
            // Check if path was dead end
            GridPos top = pos.peek();
113
114
115
            if (top.m_row != -1 && top.m_col != -1
    ) {
                // Check if dead end is actually end
116
     of the maze
117
                if (top.m row == MAZE LENGTH - 2 &&
                     top.m_col == MAZE_LENGTH - 1) {
118
119
                     m_maze[curr_loc.m_row][curr_loc.
    m_col] = '#';
                     pos.push(curr_loc);
120
                } else {
121
122
                     // Mark dead end to prevent
    returning
                     m_maze[curr_loc.m_row][curr_loc.
123
    m_{col} = 'X';
124
                // Back track and try another route
125
                curr_loc = pos.peek();
126
127
                pos.pop();
```

```
File - C:\Users\Owen Jones\Desktop\School Files\Winter2023\PROG2400 Files\assignment-2-OwenJRJones\src\MazeSolver.
128
                    continue;
129
               }
          }
130
131
132
          // Erase dead end marks from maze
133
          for (auto &i: m_maze) {
134
               for (auto &j: i) {
                    if (j == 'X') {
135
136
137
                   }
               }
138
          }
139
140
141
          // Save solution to file
142
          save_solution(path);
143 }
144
145 void MazeSolver::display_maze() {
146
          // Display maze to console
147
          for (auto &i: m_maze) {
               for (auto j: i) {
148
149
                   cout << j;</pre>
150
151
               cout << endl;
152
          }
153 }
```

```
File - C:\Users\Owen Jones\Desktop\School Files\Winter2023\PROG2400 Files\assignment-2-OwenJRJones\src\MazeSolver.
 1 #ifndef ASSIGNMENT_2_MAZESOLVER_H
 2 #define ASSIGNMENT_2_MAZESOLVER_H
 3
 4 #include <string>
 5
 6 class MazeSolver {
 7 private:
        auto static const MAZE_LENGTH{51};
 8
        auto static const MAZE_WIDTH{51};
 9
        char m_maze[MAZE_LENGTH][MAZE_WIDTH];
10
11 public:
        ~MazeSolver();
12
13
        void read_file(std::string &);
14
15
        void save_solution(std::string &);
16
17
18
        void solve(std::string &);
19
        void display_maze();
20
21 };
22
23 #endif //ASSIGNMENT_2_MAZESOLVER_H
24
```

```
File-C:\Users\Owen Jones\Desktop\School_Files\Winter2023\PROG2400_Files\assignment-2-OwenJRJones\src\Node.h

1 #ifndef ASSIGNMENT_2_NODE_H
2 #define ASSIGNMENT_2_NODE_H
3
4 #include "GridPos.h"
5
6 struct Node {
7    GridPos m_location;
8    Node *m_next{nullptr};
9 };
10
11 #endif //ASSIGNMENT_2_NODE_H
```

```
1 #include <iostream>
 2 #include "Stack.h"
 3
 4 using namespace std;
 5
 6 Stack::~Stack() {
       // Remove nodes using stack.pop() function
       while (m_first != nullptr) {
 8
 9
           pop();
       }
10
11 }
12
13 void Stack::push(GridPos location) {
       // Add node to stack
14
15
       auto node = new Node();
16
       node->m_location = location;
17
       node->m_next = m_first;
18
       m_first = node;
19 }
20
21 void Stack::pop() {
       // Pop top node off stack
22
       if (m_first == nullptr) {
23
24
           return;
25
       }
       auto node = m_first;
26
27
       m_first = m_first->m_next;
28
       delete node;
29 }
30
31 GridPos Stack::peek() {
32
       // Peek at top node on stack
       if (m_first == nullptr) return {-1, -1};
33
       return m_first->m_location;
34
35 }
36
37 std::ostream &operator<<(std::ostream &output,
   Stack &stack) {
```

```
// Output stack coordinates in X , Y
38
  formatted table
39
       auto node = stack.m_first;
       output << " X , Y" << endl << "+----+" <<
40
   endl;
       while (node != nullptr) {
41
42
           output << " " << node->m_location.m_col
    << " , ";
43
           output << node->m_location.m_row << endl;</pre>
44
           node = node->m_next;
45
       }
46
       output << "+----+";
47
      return output;
48 }
49
```

```
File - C:\Users\Owen Jones\Desktop\School_Files\Winter2023\PROG2400_Files\assignment-2-OwenJRJones\src\Stack.h
 1 #ifndef ASSIGNMENT_2_STACK_H
 2 #define ASSIGNMENT_2_STACK_H
 3
 4 #include "Node.h"
 5
 6 class Stack {
 7 private:
        Node *m_first{nullptr};
 8
 9 public:
        ~Stack();
10
11
12
        void push(GridPos);
13
        void pop();
14
15
16
        GridPos peek();
17
18
        friend std::ostream &operator<<(std::ostream</pre>
     &, Stack &);
19 };
20
21 #endif //ASSIGNMENT_2_STACK_H
22
```