

```
1 #ifndef ASSIGNMENT_2_GRIDPOS_H
2 #define ASSIGNMENT_2_GRIDPOS_H
3
4 struct GridPos {
5     int m_row{-1};
6     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;
5
6 int main(int argc, char **argv) {
7
8     MazeSolver maze{};
9     string file_name;
10
11     // Prompt user for filename if one was not
given, then open/read file
12     if (argc < 2) {
13         cout << "No file path was given, please"
14         specify: ";
15         getline(cin, file_name);
16     } else {
17         file_name = argv[1];
18     }
19
20     // Solve maze and display output to console
21     maze.solve(file_name);
22     maze.display_maze();
23
24     return 0;
25 }
```

```

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) {
12     fstream target_file(path);
13     string line;
14     auto count{0};
15
16     // Open file and read in maze
17     try {
18         if (target_file.is_open()) {
19             // Populate Maze 2D array with file
20             data
21             while (getline(target_file, line)) {
22                 for (auto i = 0; i < line.length
23                     ); i++) {
24                     m_maze[count][i] = line[i];
25                 }
26                 count++;
27             }
28         } catch (exception &e) {
29             cout << e.what() << endl;
30         }
31         target_file.close();
32     }
33
34 void MazeSolver::save_solution(string &path) {
35     // Save solution to file
36     fstream target_file;
37     filesystem::path p(path);

```

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

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

```

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

```

```
128         continue;
129     }
130 }
131
132 // Erase dead end marks from maze
133 for (auto &i: m_maze) {
134     for (auto &j: i) {
135         if (j == 'X') {
136             j = ' ';
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) {
148         for (auto j: i) {
149             cout << j;
150         }
151         cout << endl;
152     }
153 }
```

```
1 #ifndef ASSIGNMENT_2_MAZESOLVER_H
2 #define ASSIGNMENT_2_MAZESOLVER_H
3
4 #include <string>
5
6 class MazeSolver {
7 private:
8     auto static const MAZE_LENGTH{51};
9     auto static const MAZE_WIDTH{51};
10    char m_maze[MAZE_LENGTH][MAZE_WIDTH];
11 public:
12     ~MazeSolver();
13
14     void read_file(std::string &);
15
16     void save_solution(std::string &);
17
18     void solve(std::string &);
19
20     void display_maze();
21 };
22
23 #endif //ASSIGNMENT_2_MAZESOLVER_H
24
```



```
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
12
```

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

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

```
1 #ifndef ASSIGNMENT_2_STACK_H
2 #define ASSIGNMENT_2_STACK_H
3
4 #include "Node.h"
5
6 class Stack {
7 private:
8     Node *m_first{nullptr};
9 public:
10     ~Stack();
11
12     void push(GridPos);
13
14     void pop();
15
16     GridPos peek();
17
18     friend std::ostream &operator<<(std::ostream
    &, Stack &);
19 };
20
21 #endif //ASSIGNMENT_2_STACK_H
22
```