```
 1 #ifndef _STACK_H
 2 #define _STACK_H
 3
 4 #include "coordinate.h"
 5 #include <iostream>
 6
 7 class Node {
 8 public:
 9     Coordinate m_data {-1, -1};
10     Node* m_next {nullptr};
11 };
12
13 class Stack {
14 private:
15     Node* m_top {nullptr};
16
17 public:
18     Stack() = default;
19     virtual ~Stack();
20
21     void push(Coordinate num);
22     Coordinate peek();
23     void pop();
24     int count();
25
26     friend std::ostream& operator<<(std::ostream&
   output, Stack& stack);
27 };
28
29 std::ostream& operator<<(std::ostream& output, Stack
   & stack);
30
31 #endif
```

```cpp
#include "coordinate.h"
#include "stack.h"
#include "stack.cpp"
#include "coordinate.cpp"
#include <iostream>
#include <fstream>

using namespace std;

int main(int argc, char **argv) {

    Stack stack;
    string dir = "C:\\Users\\neilm\\OneDrive\\Documents\\School\\Neil\\assignment-2-neilmcdonald72\\tests\\";
    string diroutput = "C:\\Users\\neilm\\OneDrive\\Documents\\School\\Neil\\assignment-2-neilmcdonald72\\solved\\";
    string filename = argv[1];
    //string filename = "test2.txt";
    char maze[51][51];

    //starting position
    int row = 1;
    int col = 0;

    //open file and read to 2D array in memory
    ifstream file;
    file.open(dir + filename);

    while (!file.eof()) {
        for (int i = 0; i < 51; i++) {
            string line;
            getline(file, line);
            for (int x = 0; x < 51; x++) {
                maze[i][x] = line[x];
            }
        }
    }
    file.close();

```

```
38      //fix issue with test2 having empty spaces
39      if(filename == "test2.txt")
40      {
41          maze[49][49] = ' ';
42          maze[49][50] = ' ';
43      }
44
45
46      //mark starting position and push to stack
47      maze[row][col] = '#';
48      stack.push({row,col});
49
50      //begin solve
51      while(stack.peek().m_row != 49 || stack.peek().
    m_col != 50 ) {
52
53          //check north
54          if (maze[row-1][col] == ' '){
55              row = row - 1;
56              maze[row][col] = '#';
57              stack.push({row,col});
58          }
59          //check south
60          else if (maze[row+1][col] == ' '){
61              row = row + 1;
62              maze[row][col] = '#';
63              stack.push({row,col});
64          }
65          //check east
66          else if (maze[row][col+1] == ' '){
67              col = col + 1;
68              maze[row][col] = '#';
69              stack.push({row,col});
70          }
71          //check west
72          else if (maze[row][col-1] == ' '){
73              col = col - 1;
74              maze[row][col] = '#';
75              stack.push({row,col});
76          }
77          else {
```

```
 78             stack.pop();
 79             col = stack.peek().m_col;
 80             row = stack.peek().m_row;
 81             continue;
 82         }
 83 
 84     }
 85 
 86     //clear the maze
 87     for (int k = 0; k < 51; k++)
 88     {
 89         for (int n = 0; n < 51; n++)
 90         {
 91             if(maze[k][n] == '#')
 92             {
 93                 maze[k][n] = ' ';
 94             }
 95         }
 96     }
 97 
 98     //use stack to populate maze solution
 99     int stackSize = stack.count();
100     for (int r = 0; r < stackSize; r++)
101     {
102         maze[stack.peek().m_row][stack.peek().m_col
    ] = '#';
103         stack.pop();
104     }
105 
106 //
107 //    for (int m = 0; m < 51; m++)
108 //    {
109 //        for (int j = 0; j < 51; j++)
110 //        {
111 //            cout<< maze[m][j];
112 //        }
113 //        cout<<"\n";
114 //    }
115 
116     //save the solved maze to file
117     ofstream myfile;
```

```
118     myfile.open (diroutput + filename);
119     for (int h = 0; h < 51; h++)
120     {
121         for (int g = 0; g < 51; g++)
122         {
123             myfile << maze[h][g];
124         }
125         myfile<<"\n";
126     }
127     myfile.close();
128 
129     return 0;
130 }
```

```cpp
#include "stack.h"
#include <iostream>

Stack::~Stack() {
}

void Stack::push(Coordinate num) {

    auto new_node = new Node();
    new_node->m_data = num;
    new_node->m_next = m_top;
    m_top = new_node;
}

Coordinate Stack::peek() {
    if (m_top != nullptr)
        return m_top->m_data;
    return {-1, -1};
}

void Stack::pop() {
    if (m_top != nullptr) {
        auto node = m_top;
        m_top = m_top->m_next;
        delete node;
    } else {
        std::cout << "There are no more nodes to remove..." << std::endl;
    }
}

int Stack::count()
{
    Node* temp = m_top;
    int counter = 0;
    while (temp != nullptr) {
        temp = temp->m_next;
        counter++;
    }
    return counter;
}
```

```
41
42
43 std::ostream& operator<<(std::ostream& output, Stack
   & stack) {
44     auto node = stack.m_top;
45     while (node != nullptr) {
46         output << node->m_data << " ";
47         node = node->m_next;
48     }
49     return output;
50 }
```

```
 1 #ifndef _COORDINATE_H
 2 #define _COORDINATE_H
 3
 4 #include <iostream>
 5
 6 class Coordinate {
 7 public:
 8     int m_row;
 9     int m_col;
10
11     Coordinate(int, int);
12
13     bool operator!=(Coordinate coordinate);
14 };
15
16 std::ostream& operator<<(std::ostream& output,
   Coordinate coordinate);
17
18 #endif
```

```cpp
#include "coordinate.h"

Coordinate::Coordinate(int row, int col)
        : m_row(row)
        , m_col(col) {
}

bool Coordinate::operator!=(Coordinate coordinate) {
    return (this->m_row != coordinate.m_row) || (this->m_col != coordinate.m_col);
}


std::ostream& operator<<(std::ostream& output, Coordinate coordinate) {
    output << "Coordinate: " << coordinate.m_row << "/" << coordinate.m_col;
    return output;
}
```