```
1  //Main File to Run Program
2  #include <sstream>
3
4  #include "solver.h"
5
6  using namespace std;
7
8  int main() {
9      //Create new Solver & solve maze
10     auto solver = new Solver();
11
12     solver->load_maze("../tests/test3.txt");
13
14     solver->solve_maze();
15
16     solver->write_maze("../solved/testsolution3.txt");
17 }
```

```
1 //Header file for Solver Class
2 #include <string>
3 #include <vector>
4 #include "cursor.h"
5 #include "stack.h"
6
7 #ifndef ASSIGNMENT_2_SOLVER_H
8 #define ASSIGNMENT_2_SOLVER_H
9
10
11 class Solver {
12 public:
13     Solver() = default;
14
15     std::vector<std::string> maze;
16
17     void load_maze(std::string fileName);
18     void solve_maze();
19     void clear_maze();
20     void write_maze(std::string fileName);
21 };
22
23 #endif //ASSIGNMENT_2_SOLVER_H
24
```

```cpp
//C++ File for Solver Class
#include <iostream>
#include <fstream>
#include "solver.h"

using namespace std;

//Function to load maze from file
void Solver::load_maze(string fileName) {
    ifstream inFile;
    inFile.open(fileName);

    if (!inFile.fail()){
        string line;
        //Get each line of the maze and push it to a vector
        while (getline(inFile, line)) {
            if (!line.empty()){
                maze.push_back(line);
            }
        }
    }
    else {
        cout << "No file found." << endl;
    }
}

//Function to solve maze
void Solver::solve_maze() {
    auto cursor = new Cursor();
    auto stack = new Stack();
    bool done = false;

    //Until the cursor reaches the end of the maze, continue the pattern
    while (!done){
        //Set hash char at current cursor position
        maze[cursor->get_x_pos()][cursor->get_y_pos()] = '#';

        //Check above cursor for space
        if (maze[cursor->get_x_pos()][cursor->get_y_pos() - 1] == ' ') {
            //If space exists, set cursor data to new space
            cursor->set_y_pos(cursor->get_y_pos() - 1);
            //Push cursor data to the stack
            stack->push(*cursor);
        }
        //Check right of cursor for space
        else if (maze[cursor->get_x_pos() + 1][cursor->get_y_pos()] ==
' ') {
            cursor->set_x_pos(cursor->get_x_pos() + 1);
            stack->push(*cursor);
```

```
49          }
50          //Check below cursor for space
51          else if (maze[cursor->get_x_pos()][cursor->get_y_pos() + 1] ==
    ' ') {
52              cursor->set_y_pos(cursor->get_y_pos() + 1);
53              stack->push(*cursor);
54          }
55          //Check left of cursor for space
56          else if (maze[cursor->get_x_pos()- 1][cursor->get_y_pos()] == ' '
    ) {
57              cursor->set_x_pos(cursor->get_x_pos() - 1);
58              stack->push(*cursor);
59          }
60          //If no spaces available, pop the last node off the stack and set
     cursor data to prev. node
61          else {
62              stack->pop();
63              *cursor = stack->peek();
64          }
65
66          //Check for end condition
67          if (cursor->get_x_pos() == 49 && cursor->get_y_pos() == 50) {
68              done = true;
69          }
70      }
71
72      //Clear all hash chars from maze
73      clear_maze();
74
75      //For each node data, set an x at the data position
76      maze[1][0] = 'x';
77      while (stack->peek() != Cursor(-1, -1)) {
78          maze[stack->peek().get_x_pos()][stack->peek().get_y_pos()] = 'x';
79          stack->pop();
80      }
81  }
82
83  //Function to clear hash char from maze vector
84  void Solver::clear_maze() {
85      for (int i = 0; i < maze.size(); i++) {
86          for (int j = 0; j < maze[i].size(); j++){
87              if (maze[i][j] == '#') {
88                  maze[i][j] = ' ';
89              }
90          }
91      }
92  }
93
94  //Function to write solved maze to new file
```

```
 95 void Solver::write_maze(string filePath) {
 96     ofstream outFile;
 97     outFile.open(filePath, ios::app);
 98     if (!outFile.fail()) {
 99         for (int i = 0; i < maze.size(); i++) {
100             outFile << maze[i] << endl;
101         }
102     } else {
103         cout << "File save error" << endl;
104     }
105     outFile.close();
106 }
```

```
1  //Header File for Cursor Class
2  #ifndef ASSIGNMENT_2_CURSOR_H
3  #define ASSIGNMENT_2_CURSOR_H
4
5  #include <iostream>
6
7  class Cursor {
8  public:
9      int x_pos;
10     int y_pos;
11
12     Cursor();
13     Cursor(int x_pos, int y_pos);
14
15     int get_x_pos() const;
16     void set_x_pos(int x_pos_);
17
18     int get_y_pos() const;
19     void set_y_pos(int y_pos_);
20
21     bool operator!=(Cursor cursor);
22 };
23
24 #endif //ASSIGNMENT_2_CURSOR_H
```

```cpp
//C++ File for Cursor Class
#include "cursor.h"

using namespace std;

//Constructors
Cursor::Cursor() { x_pos = 1, y_pos = 0; };
Cursor::Cursor(int x_pos, int y_pos) : x_pos(x_pos), y_pos(y_pos) {};

//Overloaded != operator
bool Cursor::operator!=(Cursor cursor) {
    return(this->x_pos != cursor.x_pos) && (this->y_pos != cursor.y_pos);
}

//Getters & Setters
int Cursor::get_x_pos() const { return x_pos; };
void Cursor::set_x_pos(int x_pos_) { x_pos = x_pos_; }

int Cursor::get_y_pos() const { return y_pos; }
void Cursor::set_y_pos(int y_pos_) { y_pos = y_pos_; };
```

```
1  //Header File for Stack Class
2  #ifndef ASSIGNMENT_2_STACK_H
3  #define ASSIGNMENT_2_STACK_H
4
5  #include <iostream>
6  #include <vector>
7  #include "cursor.h"
8
9  class Node {
10 public:
11     Cursor m_data;
12     Node *m_next{nullptr};
13 };
14
15 class Stack {
16 private:
17     Node* m_last {nullptr};
18
19 public:
20     Stack() = default;
21     virtual ~Stack();
22
23     void push(Cursor position);
24     Cursor peek();
25     void pop();
26 };
27
28 #endif //ASSIGNMENT_2_STACK_H
```

```cpp
//C++ File for Stack Class
#include "stack.h"

using namespace std;

Stack::~Stack() {
    while(peek() != Cursor(-1, -1)){
        pop();
    }
}

//Function to add a node to the stack
void Stack::push(Cursor cursor) {
    auto new_node = new Node();
    new_node->m_data = cursor;
    //If no nodes in stack yet
    if (m_last == nullptr) {
        m_last = new_node;
    }
    else {
        new_node->m_next = m_last;
        m_last = new_node;
    }
}

//Function to peek at data of last node in stack
Cursor Stack::peek() {
    if (m_last != nullptr) {
        return m_last->m_data;
    }
    else {
        return {-1, -1};
    }
}

//Function to pop last node off the stack
void Stack::pop() {
    if (m_last != nullptr) {
        auto node = m_last;
        m_last = node->m_next;
        delete node;
    }
}
```