```
/*
This is a programming assignment that I did in my Intro to Programming class where I used queues to find the
shortest path of a maze
*/

#include <iostream>
#include "mazeio.h"
#include "queue.h"

using namespace std;

// Prototype for maze_search
int maze_search(char**, int, int);

// main function to read, solve maze, and print result
int main(int argc, char* argv[]) {
   int rows, cols, result;
   char** mymaze=NULL;

   if(argc < 2)
   {
       cout << "Please provide a maze input file" << endl;
       return 1;
   }

   mymaze = read_maze(argv[1], &rows, &cols);

   if (mymaze == NULL) {
      cout << "Error, input format incorrect." << endl;
      return 1;
   }


   result = maze_search(mymaze, rows, cols);


   if (result == 1) { // path found!
      print_maze(mymaze, rows, cols);
   }
   else if (result == 0) { // no path :(
      cout << "No path could be found!" << endl;
   }
   else { // result == -1
      cout << "Invalid maze." << endl;
   }

   for (int i = 0; i < rows; i++) {
      delete[] mymaze[i];
   }
   delete[] mymaze;



   return 0;
}

/*************************************************
 * Attempt to find shortest path and return:
 *  1 if successful
 *  0 if no path exists
 * -1 if invalid maze (not exactly one S and one F)
 *
 * If path is found fill it in with '*' characters
 *  but don't overwrite the 'S' and 'F' cells
 * NOTE: don't forget to deallocate memory in here too!
 *************************************************/
int maze_search(char** maze, int rows, int cols)
{
  // *** You complete **** CHECKPOINT 4
  int scounter = 0;
  int fcounter = 0;
  Location start;
  Location finish;
  for (int i = 0; i < rows; i ++) {
     for (int j = 0; j < cols; j++) {
      char check = maze[i][j];
      if (check == 'S') {
         scounter++;
         start.row = i;
         start.col = j;
      }
      if (check == 'F') {
         fcounter++;
         finish.row = i;
         finish.col = j;
      }
   }
}
if (scounter != 1 || fcounter !=1) {
   return -1;
}

Queue q(rows*cols);

int **explored = new int*[rows];
for ( int i = 0; i < rows; i++) {
   explored[i] = new int[cols];
}

for (int i = 0; i< rows; i++) {
  for (int j = 0; j < cols; j++) {
    explored[i][j] = 0;
  }
}



Location **predecessor = new Location *[rows];
for (int i = 0; i < rows; i++) {
   predecessor[i] = new Location[cols];
}


 explored[start.row][start.col] = 1;
 q.add_to_back(start);

 while(!q.is_empty()) {
    Location loc = q.remove_from_front();

    Location north ;
    north.row = loc.row- 1;
    north.col = loc.col;

    Location west = loc;
    west.col -= 1;
    Location south = loc;
    south.row += 1;
    Location east = loc;
    east.col += 1;

    if (north.row >= 0 && north.row < rows && north.col >= 0 && north.col < cols) {
       if (maze[north.row][north.col] != '#' && explored[north.row][north.col] != 1) {
          explored[north.row][north.col] = 1;
          q.add_to_back(north);
          predecessor[north.row][north.col] = loc;
       }
    }

    if (west.row >= 0 && west.row < rows && west.col >= 0 && west.col < cols) {
        if (maze[west.row][west.col] != '#' && explored[west.row][west.col] != 1) {
           explored[west.row][west.col] = 1;
           q.add_to_back(west);
           predecessor[west.row][west.col] = loc;
        }
  }

    if (south.row >= 0 && south.row < rows && south.col >= 0 && south.col < cols) {
        if (maze[south.row][south.col] != '#' && explored[south.row][south.col] != 1) {
           explored[south.row][south.col] = 1;
           q.add_to_back(south);
```

```
             predecessor[south.row][south.col] = loc;
          }
    }

      if (east.row >= 0 && east.row < rows && east.col >= 0 && east.col < cols) {
          if (maze[east.row][east.col] != '#' && explored[east.row][east.col] != 1) {
             explored[east.row][east.col] = 1;
             q.add_to_back(east);
             predecessor[east.row][east.col] = loc;
          }
    }

   }

   int end = explored[finish.row][finish.col];

   if (end == 1) {
     Location fin = predecessor[finish.row][finish.col];

     while (maze[fin.row][fin.col] != 'S') {
       maze[fin.row][fin.col] = '*';
       fin = predecessor[fin.row][fin.col];
      }
   }

   for (int i = 0; i < rows; i++) {
    delete[] explored[i];
   }
   delete[] explored;

   for (int i = 0; i < rows; i++) {
    delete[] predecessor[i];
   }
   delete[] predecessor;

   if(end == 1) {
      return 1;
   } else {
      return 0;
   }

}
```