Mouse in a Maze Solver Using Stacks

Ibrahim Hamada Ibrahim Mohamed

Contact info:
ibrahimhamada439@gmail.com

A Stack Data Structure will be used to store the spots where the mouse goes. The nodes that are stored in the stack is a struct that holds the following variables.

```
struct node {
	int x; //x-coordinate of the node
	int y; //y-coordinate of the node
	int dir = 0;
	int status = -1;
};
```

-dir variable will hold an integer which indicates the next direction that is going to be tried.
If dir=0, Up direction will be tried.
If dir=1, left direction will be tried.
If dir=2, down direction will be tried.
If dir=3, right direction will be tried.

-status variable will hold an integer which indicates the number of possible directions that determine the class of the currentSpot.
If status=-1, it is a dead end.
If status =0, it is continuing spot.
Else, it is an intersection spot.

The program will depend mainly on 2 functions. **mazeSolver()** is the first function that is used to allow the mouse to move through the maze in the white spots, search for the exit spot, and get out of the maze if it is reachable. currentSpot() is the second function that is used to determine and prints the class of that currentSpot.

- mazeSolver() function:

bool mazeSolver(int* maze[], int* visitedspots[], int* path[], node start, int fx, int fy, int R, int C)
This function returns true if the mouse reaches the exit spot and returns false if it is trapped. It takes the following parameters as an argument:

a) *maze[] (pointer to 2-D array that stores the maze).
b) * visitedspots[] (pointer to 2-D array of the visited spots).
c) * path[] (pointer to 2-D array of the solution path that leads to the exit).
d) Start (a node contains the coordinates of a valid (white) entrance spot)

e) fx, fy (the coordinates of the exit spot).
f) R, C (rows and columns of the maze).

Algorithm:

1. Defining two stacks of spot nodes s, visited. s stores the nodes of the solution path. visited stores all the visited nodes and if a spot is visited n times, it would be pushed to visited stack n times.
2. Mark the start spot as a visited spot and a spot in the solution path.
3. Push the start point to both s, visited stacks.
4. The function currentSpot**()** is called to print the status of the start node.
5. Check if s stack is empty, the function returns false which indicates that the mouse is trapped.
6. While s stack is not empty and the user wants to continue, the following steps will be executed.
7. Define a temp node and store the s.peek() in that temp node.
8. Increment the direction variable of the peek node.
9. Check if the current node is the exit node, the function returns true.
10. If it is not the exit spot, all other possible directions of the peek node will be tried one by one in an anti-clockwise manner (front, left, down, right). The direction variable of the node is incremented each iteration to determine the taken direction in the next iteration.
11. If a certain direction is taken (available) and the mouse goes to a (up, left, down, or right) spot, this spot will be marked as a visited spot and a spot in the solution path. Also, it will be pushed to both s, visited stacks.
12. If the all directions are tried and unavailable because of being black or visited spot (reach direction variable = 4), we will pop this spot node from both s, visited stacks and retrack one step back to the last spot before the current spot.
13. The function currentSpot**()** is called to print the status of the current spot.

- currentSpot() function:

void currentSpot(int* maze[], ArrayStack<node> v, int fx, int fy, int R, int C)

This function is used to print the class of that currentSpot. It takes the following parameters as an argument:

a) *maze[] (pointer to 2-D array that stores the maze).
b) v (stack of all the visited spots).
c) fx, fy (the coordinates of the exit spot).
d) R, C (rows and columns of the maze).

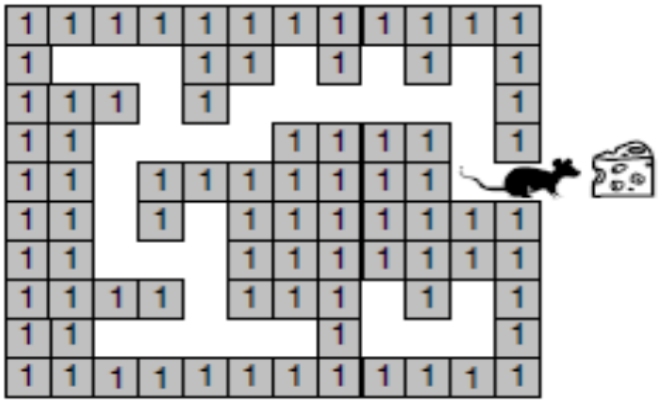
Algorithm:

1. This function will return if the v stack is empty.
2. If v is not empty, two temp nodes a, b will be defined.
3. If there is only one node in the visited stack which means that the mouse is still at the start point, we will pop this start node and count the possible (available) directions out of (up, right, left) by incrementing the status variable of that node (which is initialized to be -1).
4. If there are more than one node in the visited stack, the first two nodes will be popped from the visited stack.
5. All possible directions for that node will be counted except the direction of the last visited node before that node. The status variable (which is initialized to be -1) will be incremented till all possible directions are counted.
6. The popped nodes will be pushed again to the visited stack.
7. The status of the currentSpot will be printed.
8. If node's coordinates = coordinates of exit spot, so it the exit spot.
9. If status = -1, it is a dead end.
10. If status = 0, it is continuing spot.
11. Else, it is an intersection spot.

- **I have implemented the previous algorithm by C++ code and I will attach this code at the end of my answer.**

I have tested the given maze using the code I have implemented to make sure of the correctness of the algorithm, then I made two more mazes to test my code as required in the question.

1- The results of running the code on the given maze:



Case I: Solution found:

Enter Coordinates of the starting point: 0 0
This is a black spot! Enter Another Start Spot: 0 1
This is a black spot! Enter Another Start Spot: 2 1
This is a black spot! Enter Another Start Spot: 1 2
The Mouse Now is at Spot (1 , 2)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 1)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (1 , 2)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 3)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (3 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (5 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 4)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (5 , 4)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (6 , 4)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (7 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 4)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (8 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 2)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (8 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 4)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (8 , 5)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 6)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (8 , 5)Which Belongs to [Continuing] Class

The Mouse Now is at Spot (8 , 4)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (7 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 4)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (6 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (5 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 3)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (3 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 5)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 5)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 6)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 6)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (2 , 6)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 7)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 8)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 8)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (2 , 8)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 9)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 10)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 10)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (2 , 10)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (3 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 11)Which Belongs to [Exit] Class

Path Found!

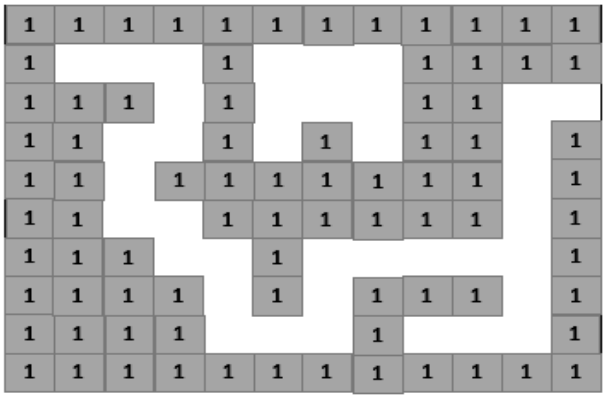
1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	0	0	1
1	1	1	0	0	0	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1

Case II: Trapped!

Enter Coordinates of the starting point: 7 11
This is a black spot! Enter Another Start Spot: 7 10
The Mouse Now is at Spot (7 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 9)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 8)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 8)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (8 , 8)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 9)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 10)Which Belongs to [Continuing] Class

No Path Found! Trapped!!!!!!

2- The results of running the code on the Maze1:



Case I: Solution found:

Enter Coordinates of the starting point: 0 0
This is a black spot! Enter Another Start Spot: 2 1
This is a black spot! Enter Another Start Spot: 4 2
The Mouse Now is at Spot (4 , 2)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (3 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (1 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (1 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (1 , 1)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (1 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (1 , 3)Which Belongs to [Continuing] Class

The Mouse Now is at Spot (2 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 2)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (5 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (5 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 5)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 6)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 6)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 6)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 7)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 8)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 9)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 10)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (5 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 11)Which Belongs to [Exit] Class

Path Found!

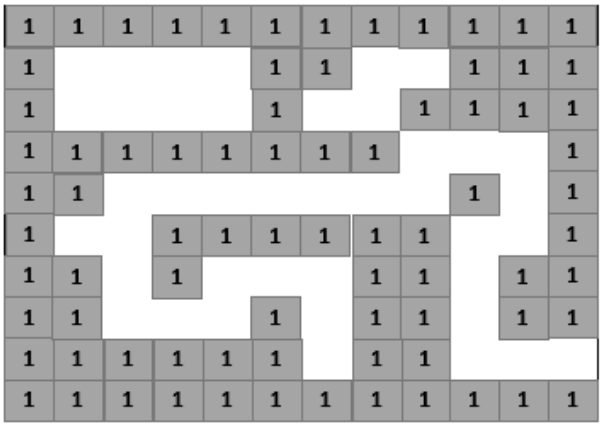
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	0	1
1	1	0	1	1	1	1	1	1	1	0	1
1	1	0	0	1	1	1	1	1	1	0	1
1	1	1	0	0	1	0	0	0	0	0	1
1	1	1	1	0	1	0	1	1	1	1	1
1	1	1	1	0	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1

Case II: Trapped!

Enter Coordinates of the starting point: 1 5
The Mouse Now is at Spot (1 , 5)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 5)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (3 , 5)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (2 , 5)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 6)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 6)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 7)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 7)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (3 , 7)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (2 , 7)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 7)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (1 , 6)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 6)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 5)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 5)Which Belongs to [Intersection] Class

No Path Found! Trapped!!!!!!

3- The results of running the code on the Maze2:



Case I: Solution found:

Enter Coordinates of the starting point: 0 0
This is a black spot! Enter Another Start Spot: 0 1
This is a black spot! Enter Another Start Spot: 4 4
The Mouse Now is at Spot (4 , 4)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (4 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (5 , 2)Which Belongs to [Intersection] Class

The Mouse Now is at Spot (5 , 1)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (5 , 2)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (6 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 5)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 6)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 6)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 6)Which Belongs to [Dead end] Class
The Mouse Now is at Spot (7 , 6)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 6)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 5)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (5 , 2)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (4 , 2)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 3)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 4)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (4 , 5)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 6)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 7)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 8)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 8)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 9)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (3 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (4 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (5 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (5 , 9)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (6 , 9)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (7 , 9)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 9)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 10)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (8 , 11)Which Belongs to [Exit] Class

Path Found!

1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	0	0	1	0	1
1	1	1	1	1	1	1	1	1	0	0	1
1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1

Case II: Trapped!

Enter Coordinates of the starting point: 1 1
The Mouse Now is at Spot (1 , 1)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 1)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 2)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 2)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 3)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 3)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (1 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 4)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (2 , 3)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 3)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (1 , 2)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 2)Which Belongs to [Intersection] Class
The Mouse Now is at Spot (2 , 1)Which Belongs to [Continuing] Class
The Mouse Now is at Spot (1 , 1)Which Belongs to [Intersection] Class

No Path Found! Trapped!!!!!!

Here is the code:

```
#include <cstring>
#include <iostream>
#include "ArrayStack.h"
using namespace std;

void currentSpot( int* maze[], ArrayStack<node> v, int fx, int fy, int R, int C) {
node a, b;
if (v.isEmpty())
return;
else {
v.pop(a);
if (!v.isEmpty()) {
	v.pop(b);
	if (a.x + 1 < R && maze[a.x + 1][a.y] == 0 && (a.x + 1 != b.x || a.y != b.y))
		a.status++;
	if (a.x - 1 >= 0 && maze[a.x - 1][a.y] == 0 && (a.x - 1 != b.x || a.y != b.y))
		a.status++;
	if (a.y + 1 < C && maze[a.x][a.y + 1] == 0 && (a.x != b.x || a.y + 1 != b.y))
		a.status++;
	if (a.y - 1 >= 0 && maze[a.x][a.y - 1] == 0 && (a.x != b.x || a.y - 1 != b.y))
		a.status++;

	v.push(b);
	v.push(a);
}
else{
	if (a.x + 1 < R && maze[a.x + 1][a.y] == 0 )
		a.status++;
	if (a.x - 1 > 0 && maze[a.x - 1][a.y] == 0 )
		a.status++;
	if (a.y + 1 < C && maze[a.x][a.y + 1] == 0 )
		a.status++;
	if (a.y - 1 > 0 && maze[a.x][a.y - 1] == 0 )
		a.status++;

	v.push(a);
}
}


cout << "The Mouse Now is at Spot ( " << a.x << " , " << a.y << " )";
if (a.x == fx && a.y == fy) {
cout << "Which Belongs to [Exit] Class" << endl;
return;
}
if (a.status == -1) {
cout << "Which Belongs to [Dead end] Class" << endl;
return;
}
else if (a.status == 0) {
cout << "Which Belongs to [Continuing] Class" << endl;
return;
}
else {
cout << "Which Belongs to [Intersection] Class" << endl;
```

```
return;
}
}


void printSolution(int* path[], int R, int C)
{
for (int i = 0; i < R; i++) {
for (int j = 0; j < C; j++) {
	cout << "    " << path[i][j];
}
cout << endl;
}
}



bool mazeSolver(int* maze[], int* visitedspots[], int* path[], node start, int fx, int
fy, int R, int C)
{
int i = start.x , j = start.y;
ArrayStack<node> s(R*C);
ArrayStack<node> visited(R * C * R);
node temp;
temp.x = i;
temp.y = j;
visitedspots[i][j] = 1;
path[i][j] = 0;
visited.push(temp);
currentSpot(maze, visited, fx, fy, R, C);
s.push(temp);

while (!s.isEmpty()) {
s.peek(temp);
int d = temp.dir;
i = temp.x;
j = temp.y;
temp.dir++;
node t;
s.pop(t);
s.push(temp);

if (i == fx && j == fy) {
	return true;
}

if (d == 0) {
	if (i - 1 >= 0 && maze[i - 1][j] == 0 && visitedspots[i - 1][j] == 0) {
		node temp1;
		temp1.x = i - 1;
		temp1.y = j;
		visitedspots[i - 1][j] = 1;
		path[i - 1][j] = 0;
		s.push(temp1);
		visited.push(temp1);
		currentSpot(maze, visited, fx, fy, R, C);
	}
```

```
}

else if (d == 1) {
	if (j - 1 >= 0 && maze[i][j - 1] ==0 && visitedspots[i][j - 1] ==0) {
		node temp1;
		temp1.x = i;
		temp1.y = j - 1;
		visitedspots[i][j - 1] = 1;
		path[i][j - 1] = 0;
		s.push(temp1);
		visited.push(temp1);
		currentSpot(maze, visited, fx, fy, R, C);
	}
}

else if (d == 2) {
	if (i + 1 < R && maze[i + 1][j] == 0 && visitedspots[i + 1][j] ==0) {
		node temp1;
		temp1.x = i + 1;
		temp1.y = j ;
		visitedspots[i + 1][j] = 1;
		path[i + 1][j] = 0;
		s.push(temp1);
		visited.push(temp1);
		currentSpot(maze, visited, fx, fy, R, C);
	}
}
else if (d == 3) {
	if (j + 1 < C && maze[i][j + 1] == 0 && visitedspots[i][j + 1] == 0) {
		node temp1;
		temp1.x = i;
		temp1.y = j + 1;
		visitedspots[i][j + 1] = 1;
		path[i][j + 1] = 0;
		s.push(temp1);
		visited.push(temp1);
		currentSpot(maze, visited, fx, fy, R, C);
	}
}
else {
	path[i][j] = 1;
	node t2;
	s.pop(t2);
	visited.pop(t2);
	currentSpot(maze, visited, fx, fy, R, C);
}
}
return false;
}


int main()
{
node start;
int R = 10; int C = 12; int fx = 4; int fy = 11;
int** visitedspot = new int* [R];
for (int i = 0; i < R; ++i)
visitedspot[i] = new int[C];
```

```
for (int i = 0; i < R; i++) {
for (int j = 0; j < C; j++) {
	visitedspot[i][j] = 0;
}
}


int** path = new int* [R];
for (int i = 0; i < R; ++i)
path[i] = new int[C];
for (int i = 0; i < R; i++) {
for (int j = 0; j < C; j++) {
	path[i][j] = 1;
}
}

int** maze = new int* [R];
for (int i = 0; i < R; ++i)
maze[i] = new int[C];

maze[0][0] = 1; maze[0][1] = 1; maze[0][2] = 1; maze[0][3] = 1; maze[0][4] = 1;
maze[0][5] = 1; maze[0][6] = 1; maze[0][7] = 1; maze[0][8] = 1; maze[0][9] = 1;
maze[0][10] = 1; maze[0][11] = 1;
maze[1][0] = 1; maze[1][1] = 0; maze[1][2] = 0; maze[1][3] = 0; maze[1][4] = 1;
maze[1][5] = 1; maze[1][6] = 0; maze[1][7] = 1; maze[1][8] = 0; maze[1][9] = 1;
maze[1][10] = 0; maze[1][11] = 1;
maze[2][0] = 1; maze[2][1] = 1; maze[2][2] = 1; maze[2][3] = 0; maze[2][4] = 1;
maze[2][5] = 0; maze[2][6] = 0; maze[2][7] = 0; maze[2][8] = 0; maze[2][9] = 0;
maze[2][10] = 0; maze[2][11] = 1;
maze[3][0] = 1; maze[3][1] = 1; maze[3][2] = 0; maze[3][3] = 0; maze[3][4] = 0;
maze[3][5] = 0; maze[3][6] = 1; maze[3][7] = 1; maze[3][8] = 1; maze[3][9] = 1;
maze[3][10] = 0; maze[3][11] = 1;
maze[4][0] = 1; maze[4][1] = 1; maze[4][2] = 0; maze[4][3] = 1; maze[4][4] = 1;
maze[4][5] = 1; maze[4][6] = 1; maze[4][7] = 1; maze[4][8] = 1; maze[4][9] = 1;
maze[4][10] = 0; maze[4][11] = 0;
maze[5][0] = 1; maze[5][1] = 1; maze[5][2] = 0; maze[5][3] = 1; maze[5][4] = 0;
maze[5][5] = 1; maze[5][6] = 1; maze[5][7] = 1; maze[5][8] = 1; maze[5][9] = 1;
maze[5][10] = 1; maze[5][11] = 1;

maze[6][0] = 1; maze[6][1] = 1; maze[6][2] = 0; maze[6][3] = 0; maze[6][4] = 0;
maze[6][5] = 1; maze[6][6] = 1; maze[6][7] = 1; maze[6][8] = 1; maze[6][9] = 1;
maze[6][10] = 1; maze[6][11] = 1;
maze[7][0] = 1; maze[7][1] = 1; maze[7][2] = 1; maze[7][3] = 1; maze[7][4] = 0;
maze[7][5] = 1; maze[7][6] = 1; maze[7][7] = 1; maze[7][8] = 0; maze[7][9] = 1;
maze[7][10] = 0; maze[7][11] = 1;
maze[8][0] = 1; maze[8][1] = 1; maze[8][2] = 0; maze[8][3] = 0; maze[8][4] = 0;
maze[8][5] = 0; maze[8][6] = 0; maze[8][7] = 1; maze[8][8] = 0; maze[8][9] = 0;
maze[8][10] = 0; maze[8][11] = 1;
maze[9][0] = 1; maze[9][1] = 1; maze[9][2] = 1; maze[9][3] = 1; maze[9][4] = 1;
maze[9][5] = 1; maze[9][6] = 1; maze[9][7] = 1; maze[9][8] = 1; maze[9][9] = 1;
maze[9][10] = 1; maze[9][11] = 1;
cout << "Enter Coordinates of the starting point: ";
cin >> start.x >> start.y;
while (maze[start.x][start.y] == 1) {
cout << "This is a black spot! Enter Another Start Spot: ";
cin >> start.x >> start.y;
}
```

```
if (mazeSolver(maze, visitedspot, path, start, fx, fy, R, C))  {
cout << "\n\nPath Found!" << '\n';
printSolution(path, R, C);

}
else
cout << "\n\nNo Path Found! Trapped!!!!!!" << '\n';


return 0;
}
```