**Lab VI Maze**

**CPSC 2150 Gladys Monagan**

You may use the STL stack or you can use your own version.

**In a maze, some cells are available and some cells are blocked. The purpose of the maze is to go from the entry cell to the exit cell (also called the goal cell) by only using the available cells that are not blocked.**

**In the 8x8 maze below, the cell with the origin (0,0) is at the top left hand corner. The first coordinate increases down and the second coordinate increases to the right. In this example, the entry cell is (6,0) and the goal cell is (7,4). A path from the entry cell to the goal cell has been drawn in with arrows. In this maze, the blocked cells are in black, the white cells are available and the path found is in light gray.**

**The coordinates of the path in reverse order are (7,4) (6,4) (6,5) (5,5) (4,5) (3,5) (3,4) (2,4) (1,4) (1,3) (1,2) (1,1) (2,1) (3,1) (3,2) (3,3) (4,3) (5,3) (5,2) (6,2) (6,1) (6,0).**

Given an NxN maze as described above, write a function path in C++ to find the path from the entry to the goal (i.e. to the exit). Output this path (in your function path) starting with the “goal cell” back to the “entry cell”. The only possible moves from one cell to the next are horizontal and vertical (there are no diagonal moves between two cells) and the cells need to be open, not blocked, for the move to happen. If there is no solution to the maze, print out an appropriate message (inside the function path). Note that the entry cell and the goal cell are not necessarily against the edge of the maze. You can write helper functions. You can print the path in a helper function called from inside of path.

**You are passed the information about the maze as a two dimensional array of cells with boolean values where**

**maze[i][j] = true means that the cell is open and can be traversed**

**maze[i][j] = false means that the cell is blocked and cannot be used**

In your function there should be backtracking when a dead end is encountered.

To store each cell of the maze in the Stack, you could use a structure

struct Cell{

int i,j;

};

and you can then represent the position of a cell with a single integer encoding (i,j) as i\*N + j

Write the function path in C++ using one of the two function prototypes given below.

The size of the maze is a global int variable N that has been initialized correctly.

Note that the array maze can be overwritten if you want to. Output the path in reverse, i.e. from goal cell to entry cell.

void path(int start\_i, int start\_j,// coordinates of the entry cell

int goal\_i, int goal\_j, // coordinates of destination cell

bool maze[][N]); // array with values for blocked

// and free cells

// maze has been declared as NxN

If you prefer, you can have a working NxN dimensional array marked passed as well. In such a case, the function prototype that you would use is:

void path(int start\_i, int start\_j,// coordinates of the entry cell

int goal\_i, int goal\_j, // coordinates of destination cell

bool maze[][N], // array with values for blocked

// and free cells

// maze has been declared as NxN

bool marked[][N]); // array with uninitialized values

// marked has been declared as NxN