

# CSE 102 Spring 2025 – Computer Programming Assignment 7

**Due on April 23, 2025 at 23:59**

In this assignment, you will implement a terminal-based version of the classic Minesweeper game. The game board will be randomly generated in size ( $N \times N$ , where  $2 \leq N \leq 10$ ), and some cells will be randomly designated as mines. The user will attempt to uncover cells while avoiding mines. The number of mines adjacent to each uncovered cell will be shown. If the user selects a cell with zero adjacent mines, all neighboring cells will automatically be revealed using a recursive flood fill. The player can also use an "undo" command to reverse their most recent move. At the end of the game, the sequence of moves will be saved in a formatted output file.

## 1. Board Generation

- At the start of the program, generate a random board size  $N$  ( $2 \leq N \leq 10$ ).
- Some cells will be randomly designated as mines.
- Save the generated map to a file called **map.txt**

## 2. Gameplay Mechanics

- The player should be able to:
  - Enter a move in the form of two integers: row col
  - Or type undo to reverse the last move
- Use a custom Stack structure to store moves
- If the player hits a mine, show “**BOOM! You hit a mine. Game Over.**”
- If the cell has 0 neighboring mines, **trigger recursive flood fill to uncover adjacent cells**
- After each move, show the current board using ASCII characters:
  - # for hidden cells
  - 0-8 for revealed cells indicating number of nearby mines
  - X for a mine that was hit

## 3. Stack

A **stack** is a data structure that follows the **Last In, First Out (LIFO)** principle. This means that the most recently added item is the first one to be removed—like a stack of plates: you add to the top, and you also remove from the top.

In the context of this assignment, the stack will be used to store the history of player moves. When the player types ‘**undo**’, the most recent move will be removed (popped) from the stack and undone on the game board.

- You are required to implement a stack: a Last-In, First-Out (LIFO) data structure.
- Since ‘**struct**’ is not covered yet, implement the stack using two **fixed-size arrays** to store row and column values separately, and an integer variable to track the top of the stack.
- Your implementation must include the following operations:

- `push`: Add a move (row and column) to the top of the stack
- `pop`: Remove the most recent move
- `isEmpty`: Check if the stack is empty

- Here is a **basic example** you can follow:

```
#define MAX_MOVES 100

int move_rows[MAX_MOVES];
int move_cols[MAX_MOVES];
int move_top = 0;

void push(int row, int col) {
    if (move_top < MAX_MOVES) {
        move_rows[move_top] = row;
        move_cols[move_top] = col;
        move_top++;
    }
}

void pop() {
    if (move_top > 0)
        move_top--;
}

int isEmpty() {
    return move_top == 0;
}
```

#### 4. Output File

When the game ends, write all valid moves to **moves.txt** in the following format:

```
--- Game Moves ---
Move 1: (Row 1, Col 1)
Move 2: (Row 2, Col 2)
...
Total Moves: N
```

#### IMPORTANT NOTES:

- Submit your homework as a zip file named as your student id (StudentID.zip) and this file should include:
  - **minesweeper.c**: Your C source code
  - **map.txt**: Automatically generated board
  - **moves.txt**: Output file after gameplay ends
  - A PDF or image with terminal screenshots showing your program running
- Programs with compilation errors will get 0.
- The output format must be as given, do not change it.
- Compile your work with given command “gcc --ansi your\_program.c -o your\_program”.

```

0 1 2 3 4 5 6
0 # # # # # #
1 # # # # # #
2 # # # # # #
3 # # # # # #
4 # # # # # #
5 # # # # # #
6 # # # # # #
Enter move (row col) or 'undo': 0 1
0 1 2 3 4 5 6
0 # 3 # # # #
1 # # # # # #
2 # # # # # #
3 # # # # # #
4 # # # # # #
5 # # # # # #
6 # # # # # #
Enter move (row col) or 'undo': 6 6
0 1 2 3 4 5 6
0 # 3 # # # #
1 # # # # # #
2 # # # # # #
3 # # # # # #
4 # # # # # #
5 # # # # # #
6 # # # # # #
Enter move (row col) or 'undo': 3 6
0 1 2 3 4 5 6
0 # 3 # # # #
1 # # # # # #
2 # # # # # 2 1
3 # # # # # 2 0
4 # # # # # 3 1
5 # # # # # #
6 # # # # # 1
Enter move (row col) or 'undo': 1 6
0 1 2 3 4 5 6
0 # 3 # # # #
1 # # # # # 1
2 # # # # # 2 1
3 # # # # # 2 0
4 # # # # # 3 1
5 # # # # # #
6 # # # # # 1
Enter move (row col) or 'undo': undo
Last move undone.
0 1 2 3 4 5 6
0 # 3 # # # #
1 # # # # # #
2 # # # # # 2 1
3 # # # # # 2 0
4 # # # # # 3 1
5 # # # # # #
6 # # # # # 1
Enter move (row col) or 'undo': 0 0
0 1 2 3 4 5 6
0 0 3 # # # #
1 # # # # # #
2 # # # # # 2 1
3 # # # # # 2 0
4 # # # # # 3 1
5 # # # # # #
6 # # # # # 1
BOOM! You hit a mine. Game Over.

```

The image shows two terminal windows side-by-side. The left window, titled 'map.txt', displays a 7x7 grid of characters. The right window, titled 'moves.txt', shows a history of moves with their coordinates.

Move	Row	Column
Move 1:	0	1
Move 2:	6	6
Move 3:	3	6
Move 4:	0	0

Total Moves: 4