CS 137: Assignment #6

Due on Friday, Nov 7, 2025, at 11:59 PM

Submit all programs using the Marmoset Submission and Testing Server located at
https://marmoset.student.cs.uwaterloo.ca/

Victoria Sakhnini

Fall 2025

Notes:

- Use the examples to guide your formatting for your output. Remember to terminate your output with a newline character.
- For this assignment, you may use any content covered until the end of Module 9.
- <math.h> is not allowed.
- Your solution may not pass marmoset tests for some of the tests due to the wrong implementation or memory leak. Therefore, I advise that even if you pass your tests and are very positive about your implementation, use Valgrind to detect memory leaks before submitting your solutions. GCC won't detect memory leaks, nor might your local compiler. Valgrind is also useful for detecting uninitialized variables accessing invalid index in an array and more.

Problem 1

Given the file `vlinteger.h`, which contains the following definitions and declarations:

```
struct linteger {
     int length;
      int *arr;     // array of digits to represent a very long positive integer, the most left digit is in index 0.
};
struct linteger *vlintegerCreate(void);
```

//Frees the memory

```
void vlintegerDestroy(struct linteger *v);
```

// Reads digits of a very long integer and stores them in `t1`.

```
void vlintegerRead(struct linteger *t1);
```

//Prints the length then prints the digits with no spaces in between, ended by `\n`.

//check the exact format and the additional text printed in the sample executions

```
void vlintegerPrint(struct linteger *t1);
```

// Returns the addition of `t1` and `t2`. No leading zeros to the left should be kept in the array.

```
struct linteger * vlintegerAdd(struct linteger *t1, struct linteger *t2);
```

Implement all the functions above in the file `vlinteger.c`

The functions `vlintegerCreate`, `vlintegerDestroy`, and `vlintegerPrint` are implemented for you in the provided files.

Submit `vlinteger.zip`, **which contains only** the files `vlinteger.c`, and `vlinteger.h`.

Notes:

- Make sure to check that there is no memory leak when you run `main`, as well as when you add more tests.
- No leading zeros in input or after completing calculations.

A sample test file is provided and shown here:

```c
#include "vlinteger.h"
#include <assert.h>
#include <stdio.h>
int main(void){
          struct linteger *int1 = vlintegerCreate();
          printf("Enter the digits separated by a space: \n");
          vlintegerRead(int1);
          vlintegerPrint(int1);
          assert(int1->arr[0] !=0);
          char c;
          scanf("%c", &c);
          struct linteger *int2 = vlintegerCreate();
          printf("Enter the digits separated by a space: \n");
          vlintegerRead(int2);
          vlintegerPrint(int2);
          assert(int2->arr[0] !=0);
          scanf("%c", &c);
          struct linteger *add = vlintegerAdd(int1, int2);
          printf("addition\n");
          vlintegerPrint(add);
          assert(add->arr[0] !=0);

          vlintegerDestroy(int1);
          vlintegerDestroy(int2);
          vlintegerDestroy(add);

 }
```

Sample1 execution:

```
Enter the digits separated by a space:
7 8 9 4 5 6 ?
length=6
789456
Enter the digits separated by a space:
3 2 1 9 9 9 2 *
length=7
3219992
addition
length=7
4009448
```

Sample2 execution:

```
Enter the digits separated by a space:
2 3 4 5 7 6 6 6 5 4 3 2 6 7 8 9 3 4 5 6 7 8 9 1 2 3 4 5 6 ?
length=29
23457666543267893456789123456
Enter the digits separated by a space:
9 9 9 9 9 9 8 7 6 5 4 4 4 4 4 4 4 4 4 3 2 1 1 1 1 8 9 7 6 5 4 3 9 8 7 6 9 8 8 8 8
6 5 4 3 2 9 8 7 *
length=49
99999987654444444432111189765439876988886543298 7
addition
length=49
99999987654444444434456956419766666334565455644 3
```

Problem 2

In this problem, we will represent banks and bank accounts with structures that contain pointers and write some functions that utilize these pointers.

We want all account information to be stored in the bank. Conversely, an account should be aware of which bank it belongs to. In other words, we want the bank to store an array of pointers to accounts, and each account should also store a pointer to the bank.

You are given the following files:

- `bank.h` that includes the data definitions and function headers along with documentation. Do NOT change this file.
- `bank.c` that includes the implementation of some of the functions to get you started to implement the rest of the functions.
- `bankmain.c` that consists of some tests to get you started. You should add more tests to cover all possible cases.

Submit `bank.zip`, **which contains only** the files `bank.c`, and `bank`.h.

// Make sure to check that there is no memory leak when you run `main`. As well when you add more tests.

A sample test file is provided and shown here:

```
1. int main(void) {
2.     printf("creating b0\n");
3.     Bank *b0 = CreateBank(1000, 100, 1.0001, 0.0001);
4.     assert(b0);
5.     printf("creating b1\n");
6.     Bank *b1 = CreateBank(2000, 300, 1.0001, 0.0001);
7.     assert(b1);
8.     printf("adding a0 to b0\n");
9.     Account *a0 = OpenAccount(b0);
10.    assert(a0);
11.    printf("adding a1 to b1\n");
12.    Account *a1 = OpenAccount(b1);
13.    assert(a1);
14.    printf("adding a2 to b1\n");
15.    Account *a2 = OpenAccount(b1);
16.    assert(a2);
17.    printf("deposit 200 in a0\n");
18.    Deposit(200, a0);
19.    ShowAccount(a0);
20.    printf("loan 100 for a0\n");
21.    TakeLoan(100, a0);
22.    ShowAccount(a0);
23.    printf("deposit 500 in a1\n");
24.    Deposit(500, a1);
25.    ShowAccount(a1);
26.    printf("deposit 400 in a2\n");
27.    Deposit(400, a2);
28.    ShowAccount(a2);
29.
30. // Deposit
```

```
31.     printf("deposit 300 in a0\n");
32.     Deposit(300, a0);
33.     printf("deposit 300 in a1\n");
34.     Deposit(300, a1);
35.     printf("deposit 100 in a2\n");
36.     Deposit(100, a2);
37.
38.     ShowBank(b0);
39.     printf("*********\n");
40.     ShowBank(b1);
41.     // Withdraw
42.     printf("withdraw 200 from a0\n");
43.     Withdraw(200, a0);
44.
45.     ShowBank(b0);
46.     printf("*********\n");
47.     ShowBank(b1);
48.     // Take Loan
49.     printf("100 loan to a0\n");
50.     TakeLoan(100, a0);
51.
52.     // Transfer
53.     printf("100 transfer from a1 to a0\n");
54.     Transfer(100, a1, a0);  // Fee applied - different banks
55.     printf("50 transfer from a1 to a2\n");
56.     Transfer(50, a1, a2);   // No fee same bank
57.
58.     ShowBank(b0);
59.     printf("*********\n");
60.     ShowBank(b1);
61.     // Pay Loan
62.     printf("collect loan payment b0\n");
63.     CollectLoanPayments(b0);
64.
65.     ShowBank(b0);
66.     printf("*********\n");
67.     ShowBank(b1);
68.
69.     Withdraw(5000, a1);
70.     ShowAccount(a1);
71.     printf("close a1\n");
72.
73.     CloseAccount(a1);
74.     ShowBank(b0);
75.     printf("*********\n");
76.     ShowBank(b1);
77.     printf("the end\n");
78.
79.     // Force Destroy
80.     ForceDestroyBank(b0);
81.     ForceDestroyBank(b1);
82.     return 0;
83. }
84.
```

The expected output is on the next page.

```
creating b0
creating b1
adding a0 to b0
adding a1 to b1
adding a2 to b1
deposit 200 in a0
Account #1:
Balance: 200.000
loan 100 for a0
Account #1:
Balance: 300.000
Loaned 100.000 with interest 1.000100
deposit 500 in a1
Account #1:
Balance: 500.000
deposit 400 in a2
Account #2:
Balance: 400.000
deposit 300 in a0
deposit 300 in a1
deposit 100 in a2
BANK:
Total money in bank: 900.000
Maximum loan offered: 100.000
Interest for loans: 1.000100
Fee for a money transfer: 0.000100
Account #1:
Balance: 600.000
Loaned 100.000 with interest 1.000100
----------------
*********
BANK:
Total money in bank: 2000.000
Maximum loan offered: 300.000
Interest for loans: 1.000100
Fee for a money transfer: 0.000100
Account #1:
Balance: 800.000
----------------
Account #2:
Balance: 500.000
----------------
withdraw 200 from a0
BANK:
Total money in bank: 900.000
Maximum loan offered: 100.000
Interest for loans: 1.000100
Fee for a money transfer: 0.000100
Account #1:
Balance: 400.000
Loaned 100.000 with interest 1.000100
----------------
*********
BANK:
Total money in bank: 2000.000
Maximum loan offered: 300.000
Interest for loans: 1.000100
Fee for a money transfer: 0.000100
Account #1:
Balance: 800.000
----------------
```

```
Account #2:
Balance: 500.000
----------------
100 loan to a0
100 transfer from a1 to a0
50 transfer from a1 to a2
BANK:
Total money in bank: 900.000
Maximum loan offered: 100.000
Interest for loans: 1.000100
Fee for a money transfer: 0.000100
Account #1:
Balance: 500.000
Loaned 100.000 with interest 1.000100
----------------
*********
BANK:
Total money in bank: 2000.010
Maximum loan offered: 300.000
Interest for loans: 1.000100
Fee for a money transfer: 0.000100
Account #1:
Balance: 649.990
----------------
Account #2:
Balance: 550.000
----------------
collect loan payment b0
BANK:
Total money in bank: 1000.010
Maximum loan offered: 100.000
Interest for loans: 1.000100
Fee for a money transfer: 0.000100
Account #1:
Balance: 399.990
----------------
*********
BANK:
Total money in bank: 2000.010
Maximum loan offered: 300.000
Interest for loans: 1.000100
Fee for a money transfer: 0.000100
Account #1:
Balance: 649.990
----------------
Account #2:
Balance: 550.000
----------------
Account #1:
Balance: 649.990
close a1
BANK:
Total money in bank: 1000.010
Maximum loan offered: 100.000
Interest for loans: 1.000100
Fee for a money transfer: 0.000100
Account #1:
Balance: 399.990
----------------
*********
BANK:
```

```
Total money in bank: 2000.010
Maximum loan offered: 300.000
Interest for loans: 1.000100
Fee for a money transfer: 0.000100
Account #1:
Inactive account
----------------
Account #2:
Balance: 550.000
----------------
the end
```

Problem 3

The good townfolks of Spooky Hollow have a problem: the woods around their town are haunted by ghosts! These ghosts are no joking matter: if you run into them, they will drain your life force and kill you! As an experienced ghost hunter, you must find all the ghosts and tell the good townsfolks which parts of the woods they can safely venture into.

When stepping into a quadrant of the haunted forest, your ghost-hunting sense will tingle and tell you exactly how many of the up to eight adjacent quadrants are haunted by a ghost. It cannot tell you, however, which ones they are.

With this information and the power of deduction, you can traverse the forest and find safe forest quadrants. You will be able to save the townfolk!

(In case you have not realized it yet: in this assignment question, you are to implement a video game!)

Anatomy of a Video Game

Generally, video games consist of three phases: *initialization*, *play loop*, and *resolution*.

1. During **initialization**, a map containing the location of all ghosts is read from input. The program has to keep track of two different maps:

- The *ghost map* (right) contains information about the location of all ghosts, i.e., the game's solution. Obviously, the player must not see this map.
- The *player map* (left) is an empty version of the *ghost map*. It represents the player's current knowledge about the game. During the *play loop*, the *player map* is populated with the results of the player's moves.

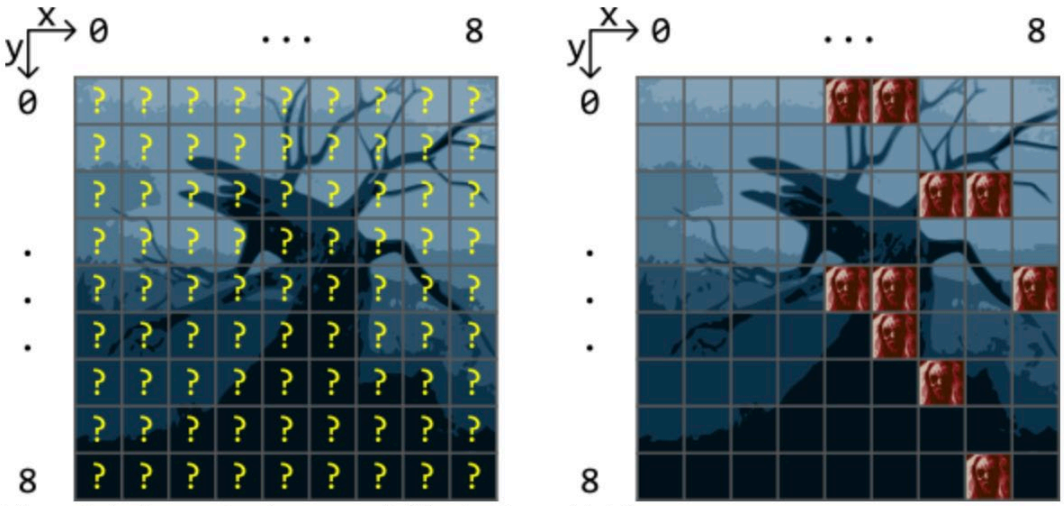

Figure 3-1: The empty *player map* (left); *ghost map* (right).

2. The **play loop** consists of an action from the player to the game, followed by output from the game to the player. The player can choose between two possible actions:

- Stepping into an unrevealed forest quadrant of the player map. The newly revealed quadrant either contains a ghost or, if it does not, the number of ghosts in all eight adjacent tiles (0 – 8). As said before, the number of ghosts is known, but their actual position is not.

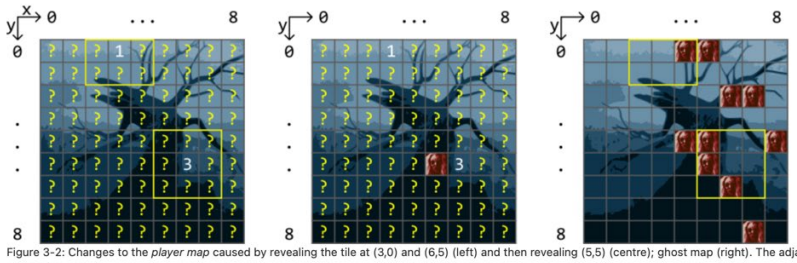

Figure 3-2: Changes to the *player map* caused by revealing the tile at (3,0) and (6,5) (left) and then revealing (5,5) (centre); ghost map (right). The adjacent tiles to the revealed ones are highlighted with yellow rectangles.

As a different example, consider revealing the tile (0,0)

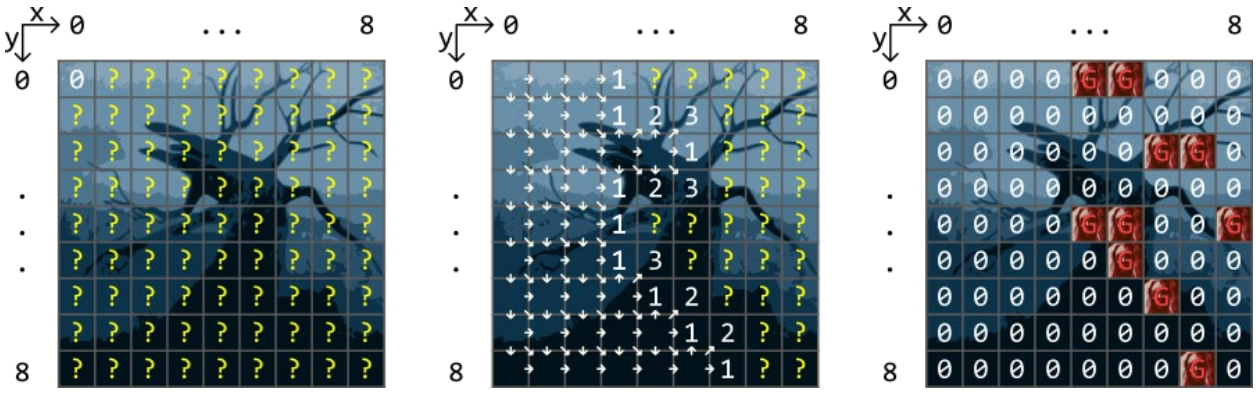

Figure 3-3: Changes to the *player map* caused by revealing the tile at (0,0) without (left) and with (centre) continued to reveal; *ghost map* (right).

In Figure 3-3 (left), we can see how the updated player map would look like. We do know, however, that all three adjacent tiles to (0,0), i.e,. (1,0), (0,1), and (1,1), are also safe to reveal. As a result, the algorithm continues revealing all neighbours of tiles not adjacent to a ghost (i.e., all neighbours of tiles that would be labelled with "0" on the player map). The white arrows in Figure 3-3 (centre) illustrate how this process of revealing spreads across the player map. All 0 have been omitted to keep a cleaner look.

- Marking one unrevealed forest quadrant or "unmarking" one marked quadrant on the player map. "Unmarking" a quadrant reverts it back to its previous state of being unrevealed. We mark a cell when we know it is a ghost to remind ourselves to avoid stepping it. When you Mark a marked cell it becomes unmarked.

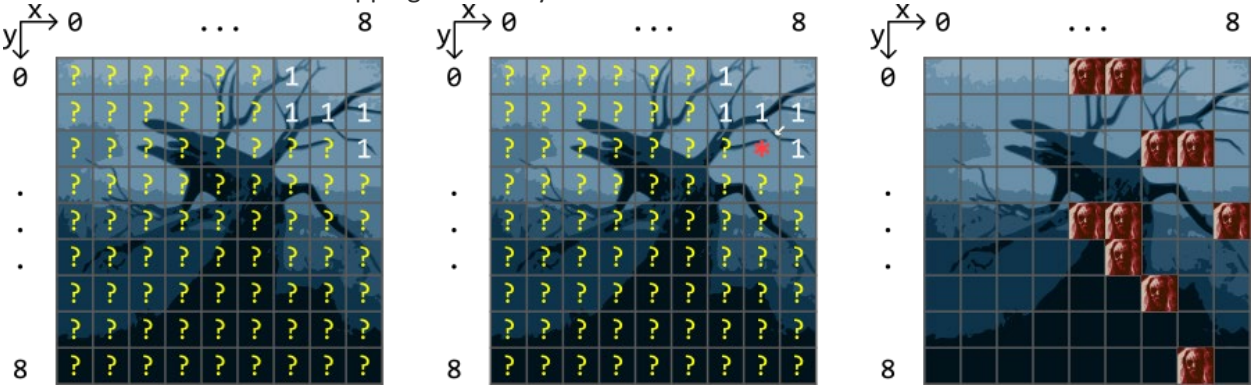

Figure 3-4: Changes to the *player map* caused by marking the tile at (7,2) (centre); *ghost map* (right). The tile at (7,2) must contain a ghost because (8,1) is adjacent to one ghost, and (7,2) was the only unrevealed neighbour to (8,1).

3. There are two conditions under which the game resolves:

- The player steps into a forest quadrant with a ghost, which ends the game with a loss for the player.
- The player successfully revealed all forest quadrants that do not contain a ghost. This ends the game with a win for the player. (Ghosts may remain partially or completely unmarked.)

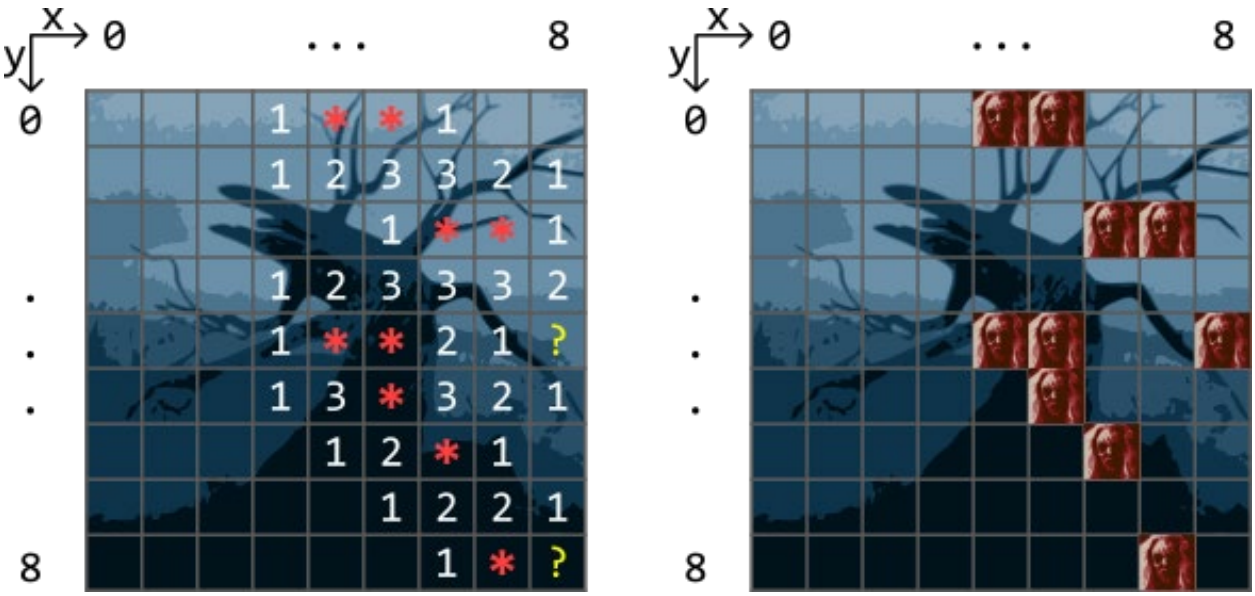

Figure 3-5: Revealing (8,8) would end the game with a win, revealing (8,4) with a loss. Marking (8,4) is not necessary to win the game. Marking (8,4) in itself will not win the game.

Implement the game. If you want to get a feel for how the game works, you can try it out online. The only notable difference, except for their lame mine-theme, is that our game does not support revealing an already revealed field ("middle-click"). Note that most of the code is given to you, you should spend time understanding the documentation and the provided code. You will need to implement only a small portion of the solution.

Note: You are to complete and submit forestgame.c containing your implemented functions along with all provided definitions and functions (including the main function. Do NOT remove the main function). **You must keep the required included libraries.**

DO NOT CHANGE MAIN FUNCTION.

We also provided *game.out* **to allow you to play the game in your student's environment and compare your program's results with the correct solution output. Note that "***The terminal will stay open for 60 seconds***." is not expected from your program (Thus do not modify the provided main function in** *foresgames.c***).**

Continue to the next page for examples:

Game 1 (User input is in bold):

Enter the width and the height: **4 3**
Enter the map row by row 0/G:
0GG0
GGGG
0GG0

```
  + 0 1 2 3 +
0 | _ _ _ _ |
1 | _ _ _ _ |
2 | _ _ _ _ |
```

STEP 0 0

```
  + 0 1 2 3 +
0 | 3 _ _ _ |
1 | _ _ _ _ |
2 | _ _ _ _ |
```

STEP 3 0

```
  + 0 1 2 3 +
0 | 3 _ _ 3 |
1 | _ _ _ _ |
2 | _ _ _ _ |
```

STEP 0 2

```
  + 0 1 2 3 +
0 | 3 _ _ 3 |
1 | _ _ _ _ |
2 | 3 _ _ _ |
```

STEP 3 2

```
  + 0 1 2 3 +
0 | 3 _ _ 3 |
1 | _ _ _ _ |
2 | 3 _ _ 3 |
```

You've found every ghosts. The town folks are safe. Well done!

Game 2 (User input is in bold):
Enter the width and the height: **5 5**
Enter the map row by row 0/G:
00000
GGGGG
00000
00000
GGGGG

```
  + 0 1 2 3 4 +
0 | _ _ _ _ _ |
1 | _ _ _ _ _ |
2 | _ _ _ _ _ |
3 | _ _ _ _ _ |
4 | _ _ _ _ _ |
```

STEP 0 0

```
  + 0 1 2 3 4 +
0 | 2 _ _ _ _ |
1 | _ _ _ _ _ |
2 | _ _ _ _ _ |
3 | _ _ _ _ _ |
4 | _ _ _ _ _ |
```

STEP 3 4

```
  + 0 1 2 3 4 +
0 | 2 _ _ _ _ |
1 | _ _ _ _ _ |
2 | _ _ _ _ _ |
3 | _ _ _ _ _ |
4 | _ _ _ G _ |
```

BOOOOOOOO! You've encountered a ghost. Game over!

Game 3 (User input is in bold):
Enter the width and the height: **4 4**
Enter the map row by row 0/G:
0GG0
0GG0
0GG0
0GG0

```
  + 0 1 2 3 +
0 | _ _ _ _ |
1 | _ _ _ _ |
2 | _ _ _ _ |
3 | _ _ _ _ |
```

STEP 3 0

```
  + 0 1 2 3 +
0 | _ _ _ 2 |
1 | _ _ _ _ |
2 | _ _ _ _ |
3 | _ _ _ _ |
```

STEP 3 1

```
  + 0 1 2 3 +
0 | _ _ _ 2 |
1 | _ _ _ 3 |
2 | _ _ _ _ |
3 | _ _ _ _ |
```

MARK 2 0

```
  + 0 1 2 3 +
0 | _ _ * 2 |
1 | _ _ _ 3 |
2 | _ _ _ _ |
3 | _ _ _ _ |
```

MARK 2 1

```
  + 0 1 2 3 +
0 | _ _ * 2 |
1 | _ _ * 3 |
2 | _ _ _ _ |
3 | _ _ _ _ |
```

STEP 1 0

```
  + 0 1 2 3 +
0 | _ G * 2 |
1 | _ _ * 3 |
2 | _ _ _ _ |
3 | _ _ _ _ |
```

BOOOOOOOO! You've encountered a ghost. Game over!