Exercise 2: 2048 Game [50 marks]

Problem Statement

2048 is a game in which the player moves numbered tiles in four possible directions (i.e., up, down, left and right) on a 4x4 grid. The objective is to merge the tiles to eventually create a tile with the number 2048. Tiles can move as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If a tile is stopped by another tile of the same value, both of them will merge into one tile with the total value of the two combined. Figure 1 shows a 2048 game in progress.

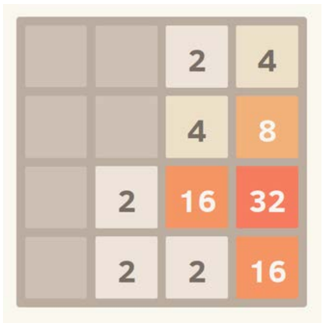

Figure 1. A 2048 game in progress

In this exercise, you are to write a program **2048.c** to simulate a simplified version of the 2048 game. In this simplified version, there are only two possible directions (i.e., up or left) for moving the tiles.

- When the move direction is **up**, the tiles at the higher row (i.e., with lower row index) are moved before the tiles at the lower row (i.e. with higher row index). For tiles in the same row, the tile on the left (i.e., lower column index) is moved before the tile on the right (i.e., higher column index).
- When the move direction is **left**, the tiles at the left column are moved before the tiles at the right column. For tiles in the same column, the tile at the top is moved before the tile at the bottom.

No new tiles are generated after each move.

For example, Figure 2a shows a sample grid with non-zero values denoting the tiles and 0s denoting spaces. If the direction of moving is up, the order of tiles chosen to move up is (0,1), (0,2), (1,0), (1,3) … and so on.

	0	1	2	3
0	0	**2**	**4**	0
1	**2**	0	0	**2**
2	**4**	0	0	**4**
3	0	**2**	**4**	0

Fig.2a. A sample 4x4 grid. Non-zero values denote tiles and 0s denote spaces.

	0	1	2	3
0	**2**	**2**	**4**	**2**
1	0	0	0	0
2	**4**	0	0	**4**
3	0	**2**	**4**	0

Fig.2b. After the tiles at (0,1), (0,2), (1,0) and (1,3) have moved up.

	0	1	2	3
0	**2**	**4**	**8**	**2**
1	**4**	0	0	**4**
2	0	0	0	0
3	0	0	0	0

Fig.2c. After the tiles at (2,0), (2,3), (3,1) and (3,2) have moved up.

Explanation of Figure 2:

> The tiles at (0,1) and (0,2) stay at the same positions since they are already at the edge of the grid, that is, they cannot move up further.
>
> Afterwards, the tiles at (1,0) and (1,3) move up and are stopped at (0,0) and (0,3) by the edge of the grid. *The status of the grid at this point is as shown in Figure 2b.*
>
> Subsequently, the tiles at (2,0) and (2,3) move up and are stopped at (1,0) and (1,3) by the two tiles at (0,0) and (0,3) respectively. Since the values of the tiles are not the same as the ones they are stopped by, no merging occurs.
>
> Lastly, the tiles at (3,1) and (3,2) move up and are stopped by two tiles at (0,1) and (0,2) of the same values respectively. Therefore, merging occurs and results in a tile of value 4 (*i.e.*, 2+2) at (0,1) and another of value 8 (*i.e.*, 4+4) at (0,2). *The final status of the grid is as shown in Figure 2c.*

The simulation starts with a given grid and follows a given sequence of directions. At the end of the simulation, a check will be performed on the grid to decide whether a target number has been formed. For example, in Figure 2c, if the target is 8, then the result of the check is *successful* because a tile of the value 8 can be found at (0,2).

Your program should read in four rows of four integers. The non-zero integers denote tiles while the 0s denote spaces in the grid. It should also read in a sequence of at most 20 characters consisting of 'U' and 'L', which represent moving up and left respectively. Lastly, it should read in an integer and check whether it is formed after moving the tiles in the given directions.

You may assume that the input is valid (*i.e.*, the integers are within the specified range and there are no illegal characters in the directions).

Write on the skeleton file **2048.c** given to you. You must include the following two functions in your program:

- **`void play(int grid[][SIZE], char directions[])`**
  which takes in a **`grid`** and a sequence of **`directions`**, and updates the grid based on the directions.
- **`int exists(int grid[][SIZE], int target)`**
  which takes in a **`grid`** and a number **`target`**, and returns 1 if **`target`** is found in the grid, or returns 0 otherwise.

You may define additional functions as needed. Check sample runs for input and output format.

Sample Runs

Five sample runs are shown below with user input highlighted in **bold**.

```
Enter grid:
0 0 0 0
0 0 0 0
0 0 0 0
1 1 1 1
Enter directions: U
Enter target: 1
1 1 1 1
0 0 0 0
0 0 0 0
0 0 0 0
1 is formed.
```

```
Enter grid:
0 0 0 1
0 0 0 1
0 0 0 1
0 0 0 1
Enter directions: L
Enter target: 2
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
2 is not formed.
```

```
Enter grid:
1 1 1 1
0 0 0 0
0 0 0 0
1 1 1 1
Enter directions: U
Enter target: 4
2 2 2 2
0 0 0 0
0 0 0 0
0 0 0 0
4 is not formed.
```

```
Enter grid:
1 0 0 1
1 0 0 1
1 0 0 1
1 0 0 1
Enter directions: L
Enter target: 2
2 0 0 0
2 0 0 0
2 0 0 0
2 0 0 0
2 is formed.
```

```
Enter grid:
2 0 0 2
0 1 1 0
0 0 0 0
1 1 1 1
Enter directions: UL
Enter target: 4
4 4 0 0
2 0 0 0
0 0 0 0
0 0 0 0
4 is formed.
```