Assignment 1: OctaFlip Game Simulation

Problem Description

Implement a simulation of an 8×8 board game called OctaFlip. Two players, Red (R) and Blue (B), take turns placing and moving their pieces according to specific rules. Your program will process a given sequence of moves and output the final state of the board or report the first invalid move.

Constraints

- You must implement your logic in C (not C++).
- Only standard libraries that can be used without the `-l` option when compiling the program are allowed.
- Your code should not produce segmentation faults for any valid/invalid input.
- Ensure bounds checking for all access to the board.
- Maintain modular and readable function definitions.

Submission Guidelines

- Submit a single `.c` file containing your implementation.
- File naming format: `hw2_YOURSTUDENTID.c`
  Example: `hw2_200012345.c`
- Upload your code via LMS before May 20, Tuesday, 11:59:59 PM.
- You are granted a total of **two days** of grace period, which can be used for either Assignment 1 or Assignment 2, or split between both.
    - For example, you may submit Assignment 1 two days late and Assignment 2 on time, or each assignment one day late. No additional extensions will be given beyond this shared grace period.

Board Encoding

The board is represented as an 8×8 grid using the following characters:

Symbol	Meaning
R	Red player's piece
B	Blue player's piece
.	Empty cell
#	Blocked cell

Game Rules

1. Turn-based Actions:
    - Players alternate turns, starting with Red.
    - On a turn, a player can clone or jump a piece.
2. Move Types:
    - Clone: Move to a directly adjacent cell (1 cell in any of the 8 directions). The original piece remains in place.
    - Jump: Move to a cell that is exactly 2 cells away (in any of the 8 directions). The original piece disappears after the move.Flip Mechanism:
    - After the move (either clone or jump), all opponent pieces in the 8 surrounding cells of the destination are flipped to the current player's color.

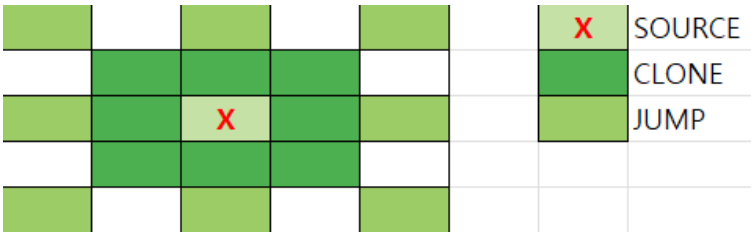

Fig1. Allowed Moves

3. Move Validity:
    - The destination cell must be . (empty).
    - The source cell must contain the current player's piece.
    - The movement must be exactly 1 or 2 steps in a valid direction (no knight or diagonal leaps beyond 2 cells).
4. Passing:
    - If a player has **no valid move**, they must pass using `0 0 0 0`.

- If **two passes occur in a row**, i.e., Red passes on turn k and then Blue immediately passes on turn k+1 (or vice versa), the game ends immediately **before any board changes**.
    i. Turn 5: Red has no moves, reads `o o o o` → pass
    ii. Turn 6: Blue has no moves, reads `o o o o` → pass
    iii. → Game ends immediately at end of turn 6.

5. Game Termination (Always executed after each player's turn)
   After **each** player's move (clone or jump) or pass, immediately check the following conditions. If **any** one of these is met, end the game and print the result:
    - There are no . (empty) cells left.
    - One player has no pieces left.
    - Both players pass consecutively.
6. Result is printed when the gamed ends
    - The winner is the player with more pieces on the board at the end.
    - If the counts are equal, the result is a Draw.

Functionality Requirements

Implement a program that:

- Parses the board and move sequence.
- Simulates the game according to the rules.
- Detects invalid moves (wrong turn, invalid jump, non-empty destination, etc.).
- **Input validation checks should be performed before the game process**
- **Even if the initial board state matches a game termination condition, you must execute at least the first turn**
- Outputs the final board state and result if all moves are valid.

Refer to the full set of sample inputs and outputs provided in `OctaFlip_examples.txt`.

There are two way to implement this assignment

1. As with other problem-solving tasks, you may **terminate** the program immediately after printing the output.
2. Alternatively, you can prompt the user to **re-enter** the string whenever the input format is invalid.

   Both approaches will be graded according to the same criteria.

Input

```
<8 lines>: initial board state (each line has 8 characters: R, B, ., or #)
<1 line> : integer N (number of moves)
<next N lines>: each with 4 integers r1 c1 r2 c2 (1 based-index)
```

Detail procedure:

- Read the 8 board lines
  - If any line is not exactly 8 characters of valid symbols (R, B, ., #), print:
  - `Board input error`
  - and terminate.
- Read N (number of turns):
  - If N is **negative, not an integer, or missing, etc.**, print:
  - `Invalid input at turn 0`
  - and terminate.
- For each turn k (1 - based index), read a group of four integers “r1 c1 r2 c2”
  - If any integer is **negative, not an integer, or missing, etc.**, print:
  - `Invalid input at turn k`
  - and terminate.

Where:

- `(r1, c1)` is the source coordinate.
- `(r2, c2)` is the destination coordinate.

Output

If an invalid move is encountered at turn `k` (1-based index), print:

```
Invalid move at turn k
```

Otherwise, print:

```
<8 lines>: final board state (each line has 8 characters: R, B, ., or #)
<1 line>: 1 line containing one of the words: ["Red", "Blue", "Draw"]
```