Assignment 5: Zebrariders

Due: 20:00, Wed 14 Apr 2021 File name: `zebrarider.cpp` Full marks: 100

Introduction

The objective of this assignment is to practice the use of 2-D arrays with loops. You will write a program related to the chess game. There are different types of pieces in chess like queen, rook, knight, etc. In some chess variants, there are additional chess piece types, e.g., *zebrarider*. It moves two vertical positions by three horizontal positions (2V3H) for an unlimited distance. It can also move 3V2H for an unlimited distance. (See Figure 1.)

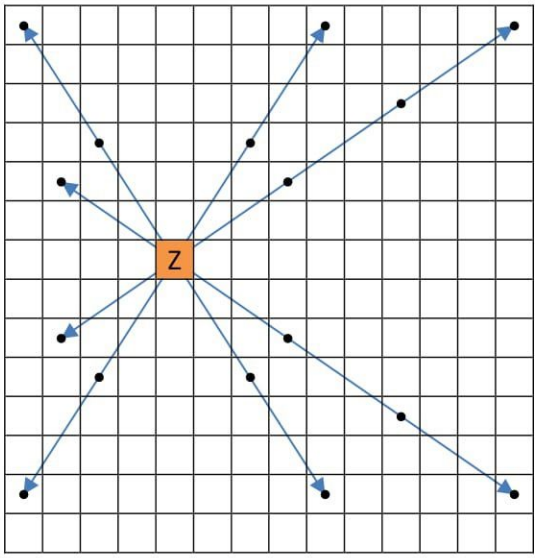

Figure 1: Possible Moves of a Zebrarider (Z) in a 14×14 Chessboard

Given an $n \times n$ chessboard, the *dominating zebrariders problem* is to put as fewest zebrariders to the board as possible so that all squares in the board are either occupied or attacked (that is, reached) by a zebrarider. In this assignment, you will write a program to put zebrariders to the board one by one until all the zebrariders "dominate" the board. Figure 2 shows an example 8×8 board configuration. In the figure, 'Z' denotes a zebrarider, '=' denotes a square that is attacked (reached) by a zebrarider, and '.' denotes an unattacked (free) square.

```
   A B C D E F G H
 0 . . Z . . . . .
 1 . . . = . . . =
 2 . . = . . = . .
 3 = . . . = . . .
 4 . . . . . Z . .
 5 . . . . . . . .
 6 . . = . . . = .
 7 . . . = . . . =
Zebrariders: 2
Attacked squares: 10
Free squares: 52
```

Figure 2: An Example 8×8 Board Configuration

Program Specification

This section describes the chessboard representation, program flow, and some special requirements.

Chessboard Representation

The chessboard will be represented by a two-dimensional array of `char`. The array elements should be either `'Z'` (zebrarider), `'='` (attacked), or `'.'` (free/unattacked).

```
const int SIZE = 8;     // Global named constant
…
    char board[SIZE][SIZE];     // Local variable
```

When `SIZE = 8`, the array elements `board[0][0]`, `board[0][SIZE-1]`, `board[SIZE-1][0]`, and `board[SIZE-1][SIZE-1]` denote the four corner squares A0, H0, A7, and H7 respectively.

Program Flow

1. The program starts with a completely free board.
2. Then the program prompts the user to enter an input position to put the zebrarider, which is always *a letter followed by an integer*. (E.g., F 4 means putting a zebrarider at position F4.)
3. A user input is *invalid* if: (a) the row and column are outside of the board, *or* (b) the input position is already occupied by a zebrarider, *or* (c) placing a zebrarider at the input position does not reduce the number of free squares. In case the user makes an invalid input, display a warning message and go back to Step 2.

   Note: (i) lowercase column letters are invalid; (ii) putting a zebrarider at a '=' position can be valid, as long as the zebrarider attacks some new square(s).
4. Update the chessboard by putting the zebrarider to the input position and attacking the squares along all the directions accordingly.
5. Repeat Steps 2–4 until the board contains no free squares. That is, only 'Z' and '=' remain.

Special Requirements

- *Global variables (variables declared outside any functions) are not allowed*. Nonetheless, `const` ones (e.g., `SIZE`) do not count.
- Your program should be decomposed into *at least four functions* (including `main()`). At least *two functions* should have *array parameter(s)*.
- Your program should be *scalable* to other values for the named constant `SIZE`. That is, your program should still *work normally for other board size*. When grading, we may modify your program by changing SIZE to other values (may be 1–26) for testing.

Sample Run

In the following sample run, the blue text is user input and the other text is the program printout. *More sample runs* are provided in *Blackboard*. Besides, you can try the provided sample program for other input. *Your program output should be exactly the same as the sample program* (same text, symbols, letter case, spacings, etc.).

```
   A B C D E F G H
 0 . . . = . . . =
 1 = . = . = . . .
 2 . . . . = . . .
 3 . . . = . Z = .
 4 . . = . . . . .
 5 . . = = . . Z .
 6 . . . = . . . =
 7 Z . . = . . . .
Zebrariders: 3
Attacked squares: 14
Free squares: 47
         ⋮          (Some inputs skipped. See full version in Blackboard.)
   A B C D E F G H
 0 = = = = Z . = =
 1 = Z = = = = . =
 2 = = = Z = = = Z
 3 Z = = = = Z = =
 4 Z = Z Z = = = Z
 5 = = = Z Z = Z =
 6 = = Z = = = = =
 7 Z = = Z = = = =
Zebrariders: 16
Attacked squares: 46
Free squares: 2
Put a zebrarider (col row): C 2↵
   A B C D E F G H
 0 = = = = Z = = =
 1 = Z = = = = . =
 2 = = Z Z = = = Z
 3 Z = = = = Z = =
 4 Z = Z Z = = = Z
 5 = = = Z Z = Z =
 6 = = Z = = = = =
 7 Z = = Z = = = =
Zebrariders: 17
Attacked squares: 46
Free squares: 1
Put a zebrarider (col row): G 1↵
   A B C D E F G H
 0 = = = = Z = = =
 1 = Z = = = = Z =
 2 = = Z Z = = = Z
 3 Z = = = = Z = =
 4 Z = Z Z = = = Z
 5 = = = Z Z = Z =
 6 = = Z = = = = =
 7 Z = = Z = = = =
Zebrariders: 18
Attacked squares: 46
Free squares: 0
```

```
   A B C D E F G H
 0 . . . . . . . .
 1 . . . . . . . .
 2 . . . . . . . .
 3 . . . . . . . .
 4 . . . . . . . .
 5 . . . . . . . .
 6 . . . . . . . .
 7 . . . . . . . .
Zebrariders: 0
Attacked squares: 0
Free squares: 64
Put a zebrarider (col row): G 5↵
   A B C D E F G H
 0 . . . . . . . .
 1 = . . . . . . .
 2 . . . . = . . .
 3 . . . = . . . .
 4 . . . . . . . .
 5 . . . . . . Z .
 6 . . . . . . . .
 7 . . . = . . . .
Zebrariders: 1
Attacked squares: 4
Free squares: 59
Put a zebrarider (col row): I 2↵
Invalid. Try again!
Put a zebrarider (col row): a 3↵
Invalid. Try again!
Put a zebrarider (col row): D -1↵
Invalid. Try again!
Put a zebrarider (col row): C 9↵
Invalid. Try again!
Put a zebrarider (col row): G 5↵
Invalid. Try again!
Put a zebrarider (col row): A 7↵
   A B C D E F G H
 0 . . . . . . . .
 1 = . . . = . . .
 2 . . . . = . . .
 3 . . . = . . = .
 4 . . = . . . . .
 5 . . . = . . Z .
 6 . . . . . . . .
 7 Z . . = . . . .
Zebrariders: 2
Attacked squares: 8
Free squares: 54
Put a zebrarider (col row): F 3↵
```