Assignment 4: No-Three-in-Line

Due: 20:00, Thu 8 Nov 2018 Full marks: 100

Introduction

Given an $n \times n$ square board, the *no-three-in-line* problem is to put as many pieces to the board as possible so that no three pieces can appear in a straight line, where "straight line" means a line along the horizontal —, vertical |, or diagonal ╲ ╱ directions. You will write a program to put pieces to the board one by one until no more possible positions can be found. Figure 1 shows an example 12×12 board configuration. In the figure, '@' denotes a piece, '.' denotes an empty square, and '=' denotes an empty square that cannot have a piece on it anymore. For example, in row 4, there are two pieces on the two ends, thus the remaining squares along the row are all '=' as we cannot have a third piece on this row anymore.

```
   A B C D E F G H I J K L
 0 = = = = @ @ = = = = = =
 1 = . . = . . . . . . . .
 2 . = = . . . . . . . . .
 3 . = @ . . . . . . . . .
 4 @ = = = = = = = = = = @
 5 . . . . = . . . . . . .
 6 . . . . . = . . . . . .
 7 . . . . . . @ . . . . .
 8 . . . . . . . = . . . .
 9 . . . . . . . . = . . .
10 . . . . . . . . . = . .
11 . . . . . . . . . . = .
```

Figure 1: An Example 12×12 Board Configuration

Program Specification

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

Board Representation

The board will be represented by a two-dimensional array of `char`. The array elements should be either `'@'` (a piece), `'.'` (empty), or `'='` (empty but cannot be put).

```
const int SIZE = 12;    // Global named constant
…
    char board[SIZE][SIZE];    // Local 2-D array
```

When `SIZE` = 12, 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, L0, A11, and L11 respectively.

Program Flow

1. The program starts with a completely empty board.
2. Then the program prompts the user to enter an input position to put a piece, which is always *a letter followed by an integer*. (E.g., `F 4` means putting a piece 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 not empty '`.`' In case the user makes an invalid input, display a warning message and go back to Step 2. Note that lowercase column letters are considered invalid.
4. Update the board by putting a piece to the input position.
5. After putting a piece, for each of the four directions — | ╲ ╱ of the input square, if there are already two pieces along that direction (that is, no pieces can be put in this line in the future), mark the remaining empty squares in that line as '='.
6. Repeat Steps 2–5 until the board contains no empty squares. That is, only '@' and '=' remain.
7. At the end, print the total number of pieces on the board.

Special Requirements

- You are *not allowed to use any global variables* in your program. (That is, do *not* declare any variables outside functions.) 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 board size other than 12*. When grading, we may modify your program by changing `SIZE` to other values (may be 1–26) for testing.

Program Output

The following shows some sample output of the program. The `blue` text is user input and the other text is the program output. You can try the provided sample program for other input. *Your program output should be exactly the same as the sample program* (i.e., same text, same symbols, same letter case, same number of spaces, etc.). Otherwise, it will be considered as *wrong*, even if you have computed the correct result.

```
   A B C D E F G H I J K L
 0 . . . . . . . . . . . .
 1 . . . . . . . . . . . .
 2 . . . . . . . . . . . .
 3 . . . . . . . . . . . .
 4 . . . . . . . . . . . .
 5 . . . . . . . . . . . .
 6 . . . . . . . . . . . .
 7 . . . . . . . . . . . .
 8 . . . . . . . . . . . .
 9 . . . . . . . . . . . .
10 . . . . . . . . . . . .
11 . . . . . . . . . . . .
Put a piece (col row): H 1↵
```

```
   A B C D E F G H I J K L
 0 . . . . . . . . . . . .
 1 . . . . . . . @ . . . .
 2 . . . . . . . . . . . .
 3 . . . . . . . . . . . .
 4 . . . . . . . . . . . .
 5 . . . . . . . . . . . .
 6 . . . . . . . . . . . .
 7 . . . . . . . . . . . .
 8 . . . . . . . . . . . .
 9 . . . . . . . . . . . .
10 . . . . . . . . . . . .
11 . . . . . . . . . . . .
Put a piece (col row): M 1↵
Invalid. Try again!
Put a piece (col row): E -3↵
Invalid. Try again!
Put a piece (col row): a 4↵
Invalid. Try again!
Put a piece (col row): A 4↵
   A B C D E F G H I J K L
 0 . . . . . . . . . . . .
 1 . . . . . . . @ . . . .
 2 . . . . . . . . . . . .
 3 . . . . . . . . . . . .
 4 @ . . . . . . . . . . .
 5 . . . . . . . . . . . .
 6 . . . . . . . . . . . .
 7 . . . . . . . . . . . .
 8 . . . . . . . . . . . .
 9 . . . . . . . . . . . .
10 . . . . . . . . . . . .
11 . . . . . . . . . . . .
Put a piece (col row): H 1↵
Invalid. Try again!
Put a piece (col row): E 9↵
   A B C D E F G H I J K L
 0 . . . . . . . . . . . .
 1 . . . . . . . @ . . . .
 2 . . . . . . . . . . . .
 3 . . . . . . . . . . . .
 4 @ . . . . . . . . . . .
 5 . . . . . . . . . . . .
 6 . . . . . . . . . . . .
 7 . . . . . . . . . . . .
 8 . . . . . . . . . . . .
 9 . . . . @ . . . . . . .
10 . . . . . . . . . . . .
11 . . . . . . . . . . . .
Put a piece (col row): I 8↵
```

```
   A B C D E F G H I J K L
 0 . . . . . . . . . . . .
 1 . . . . . . . @ . . . .
 2 . . . . . . . . . . . .
 3 . . . . . . . . . . . .
 4 @ . . . . . . . . . . .
 5 . . . . . . . . . . . .
 6 . . . . . . . . . . . .
 7 . . . . . . . . . . . .
 8 . . . . . . . . @ . . .
 9 . . . . @ . . . . . . .
10 . . . . . . . . . . . .
11 . . . . . . . . . . . .
Put a piece (col row): E 2↵
   A B C D E F G H I J K L
 0 . . . . = . . . . . . .
 1 . . . . = . . @ . . . .
 2 . . . . @ . . . . . . .
 3 . . . . = . . . . . . .
 4 @ . . . = . . . . . . .
 5 . . . . = . . . . . . .
 6 . . . . = . . . . . . .
 7 . . . . = . . . . . . .
 8 . . . . = . . . @ . . .
 9 . . . . @ . . . . . . .
10 . . . . = . . . . . . .
11 . . . . = . . . . . . .
Put a piece (col row): E 5↵
Invalid. Try again!

    ⋮       (Some moves are skipped to save space. See Blackboard for full version.)

   A B C D E F G H I J K L
 0 = @ = = = = = = = @ = =
 1 = = = = = = @ @ = = = =
 2 = = = @ @ = = = = = = =
 3 = @ = = = = = = @ = = =
 4 @ = = = = = = = = . . .
 5 = = @ = = @ = = = = = =
 6 @ = = @ = = = = = = = =
 7 = = = = = = @ = = @ = =
 8 = = = = = = = @ @ = = =
 9 = = @ = @ = = = = = = =
10 = = = = = . = = = = = =
11 = = = = = = = = = = = =
Put a piece (col row): L 4↵
```

```
   A B C D E F G H I J K L
 0 = @ = = = = = = = @ = =
 1 = = = = = = @ @ = = = =
 2 = = = @ @ = = = = = = =
 3 = @ = = = = = = @ = = =
 4 @ = = = = = = = = = = @
 5 = = @ = = @ = = = = = =
 6 @ = = @ = = = = = = = =
 7 = = = = = = @ = = @ = =
 8 = = = = = = = @ @ = = =
 9 = = @ = @ = = = = = = =
10 = = = = = = = = = = = =
11 = = = = = = = = = = = =
Num of pieces: 20
```

Submission and Marking

- Your program file name should be `no3inline.cpp`. Submit the file in Blackboard (https://blackboard.cuhk.edu.hk/).
- Insert your name, student ID, and e-mail address as comments at the beginning of your source file.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- Your program should *include suitable comments as documentation*.
- *Plagiarism is strictly monitored and heavily punished if proven.* Lending your work to others is subjected to the same penalty as the copier.