Programming in C

Deadline: *20 February, 23:59*

Assignment 2a: Check!

Your task is to write a program that reads a chessboard configuration and identifies whether a king is under attack (in check). A king is in check if it is on square which can be taken by the opponent on his next move. White pieces will be represented by uppercase letters, and black pieces by lowercase letters. The white side will always be on the bottom of the board, with the black side always on the top. For those unfamiliar with chess, here are the movements of each piece:

Pawn (p or P)

Can only move straight ahead, one square at a time. However, it takes pieces diagonally, and that is what concerns you in this problem. A black pawn can only attack one square diagonally down the board, a white pawn can only attack one square diagonally up the board.

Knight (n or N)

Has an L-shaped movement shown below. It is the only piece that can jump over other pieces.

Bishop (b or B)

Can move any number of squares diagonally, either forward or backward.

Rook (r or R)

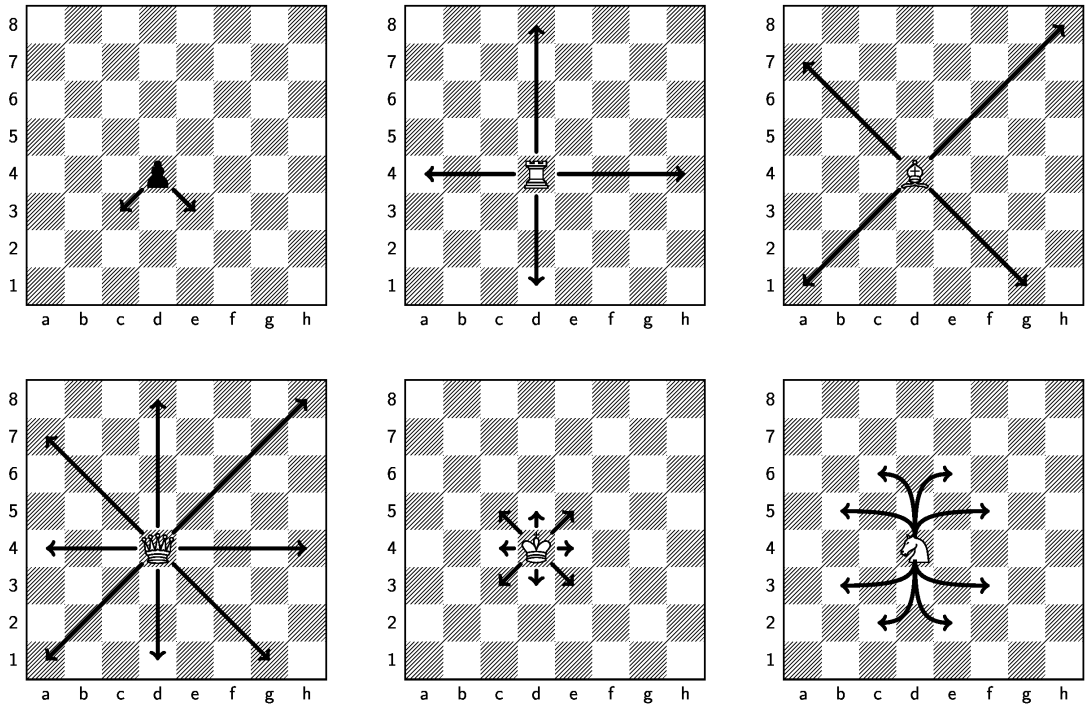
Can move any number of squares vertically or horizontally, either forward or backward.

Queen (q or Q)

Can move any number of squares in any direction (diagonally, horizontally, or vertically) either forward or backward.

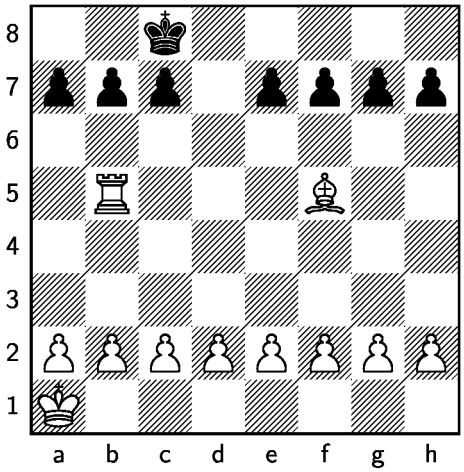
King (k or K)

Can move one square at a time in any direction (diagonally, horizontally, or vertically) either forward or backward.



Input and Output

Your program should read a board configuration from standard input. The format consists of eight lines of eight characters. A '`.`' denotes an empty square, while upper- and lowercase letters represent the pieces as defined above. For example, this board configuration:



is defined in text form as:

```
..k.....
ppp.pppp
........
.R...B..
........
........
PPPPPPPP
K.......
```

There will be no configurations where both kings are in check. A board configuration will contain exactly one white king and one black king. The output of your program should be one of the following three lines:

```
white king is in check.

black king is in check.

no king is in check.
```

Implementation tips

A 2d character array is a natural choice to store the chessboard configuration. Although not required, it is a good idea write a function to print the board to verify you have correctly read the chessboard from standard input.

Try to avoid magic numbers and literal constants. Instead use defines such as:

```
#define BSIZE       8
#define PAWN_W      'P'
#define PAWN_B      'p'
```

```
#define ROOK_W      'R'
// ..
```

or use constants where possible:

```
// BSIZE used to declare 2d array so we need #define here.
#define BSIZE       8

const char pawn_w = 'P';
const char pawn_b = 'p';
const char rook_w = 'R';
// ..
```

Think about the way you want to check for checks before you start coding. Also consider that every piece can attack in multiple directions. Implementing the function to check all those directions in a naive manner requires multiple for-loops. This kind of code is difficult to read and maintain, and it is also prone to copy-paste errors. Consider making the direction (the arrow in the pictures above) a parameter so you can reduce the number of for-loops.

Bonus Assignments

Online judge

This assignment is taken from a set of practice exercises that are used to train for programming competitions. This means you can check your implementation online with the Uva online judge system [1]. The assignment ID is 10196 and the name is "Check The Check". You need register to be able to view and submit assignments. Note that the programming challenge version is slightly different. It requires ANSI C and you should be able to process multiple board configurations. See the description on the Uva online judge system [1] for the details. As a bonus exercise you can extend your checker program such that it is accepted by the online judge.

Performance

Try to make your check finder program faster. Can you give a rough estimate of the number of moves that you need to check for an average board configuration? Can you think of another approach to find the king in check that would reduce that number? Implement this approach and compare the performance with your initial solution.

Visual improvements

In addition to reporting which (if any) king is in check, you can print the board configuration back to the screen and highlight the check in color or in some other way.

1(1, 2) http://uva.onlinejudge.org