ComS 327 Project Part 1

Checkers: read input files

Fall 2019

1 Summary

The semester-long project will be based on the board game Checkers (also known as Draughts). For the first part, you must read an input file that describes a game configuration, make sure the input file is valid, and display summary information about the input file. In later parts of the project, you will be required to do *much* more processing of the input file, and will likely need to read the game configuration into an appropriate data structure. For this part of the project, all source code must be C.

Specifically, your `Makefile` must build an executable named `info`. When run, `info` should read the input file from standard input, and write summary information to standard output. If the input file is *invalid*, then display an appropriate error message to standard error, that indicates why the input file is invalid. Otherwise, if the input file is *valid*, then the following should be written to standard output:

```
VALID INPUT
Initial configuration:
Turn: <red or black>
Red: <n> kings, <m> pawns
Black: <q> kings, <p> pawns
```

In the above, the "turn" and count of pieces should be given for the initial game configuration specified in the input file. An example input file and its associated output is given below.

2 Board coordinates

For a general overview of the rules of the game, see for example `https://simple.wikipedia.org/wiki/Checkers`. Note that there are many variants of the game, so you should refer to the project specifications to see which rules are being followed. There is also a variety of terminology. For this project, I will use the following terms:

Pawn: this is also known as a "single piece" or "man". At the beginning of a game, all pieces are pawns.

King: this is also known as a "double piece". Pawns are promoted to kings when they reach their final row.

For purposes of the project, the "standard" board along with its coordinates is shown in Figure 1. The digits 1–8 refer to the *rank* and the letters 'a'–'h' refer to the *file*. The coordinates of a board square are written as `<file><rank>`, for example, `a1`.

Pieces are placed on the black squares only, and a game normally begins with red pawns on the black squares with ranks 1–3 (`a1`, `c1`, `e1`, `g1`, `b2`, `d2`, `f2`, `h2`, `a3`, `c3`, `e3`, and `g3`) and black pawns on the black squares with ranks 6–8.

Red pawns can only increase in rank, and become kings at rank 8. Black pawns can only decrease in rank, and become kings at rank 1. Kings can both increase and decrease in rank.

Sometimes, players will set up the board incorrectly, either by placing the pawns on the *red* squares, or by rotating the board 90 degrees. Throughout the project, I will call this a *flipped* board, which means that the black squares and red squares have been exchanged as shown in Figure 2.

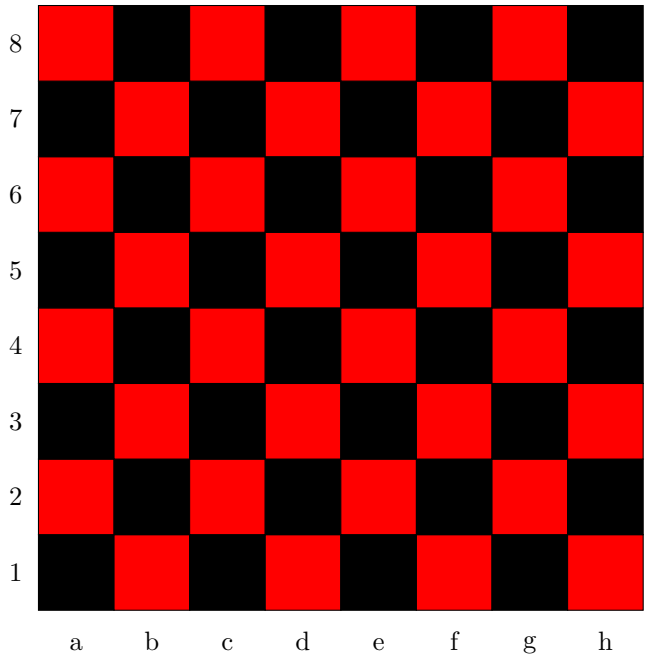

Figure 1: Standard checkerboard coordinates

3 Input file format

The input files are free-form text files. This means that any extra whitespace (spaces, tabs, carriage returns, newlines) is ignored. Also, the `'#'` character indicates that all text until the end of the current line should be ignored (i.e., for comments in the input file). The overall file format is as follows.

```
RULES:
<"capture" or "no capture">
<"single jumps" or "multiple jumps">

TURN:
<"red" or "black">

BOARD:
<see board format below>

MOVES:
<zero or more moves, format specified below>
```

During specification of the board (between `BOARD:` and `MOVES:`), the following characters are used to indicate squares on the board:

- `'"'`: an empty red square.
- `'.'`: an empty black square.
- `'b'`: a square containing a black pawn.
- `'B'`: a square containing a black king.
- `'r'`: a square containing a red pawn.
- `'R'`: a square containing a red king.

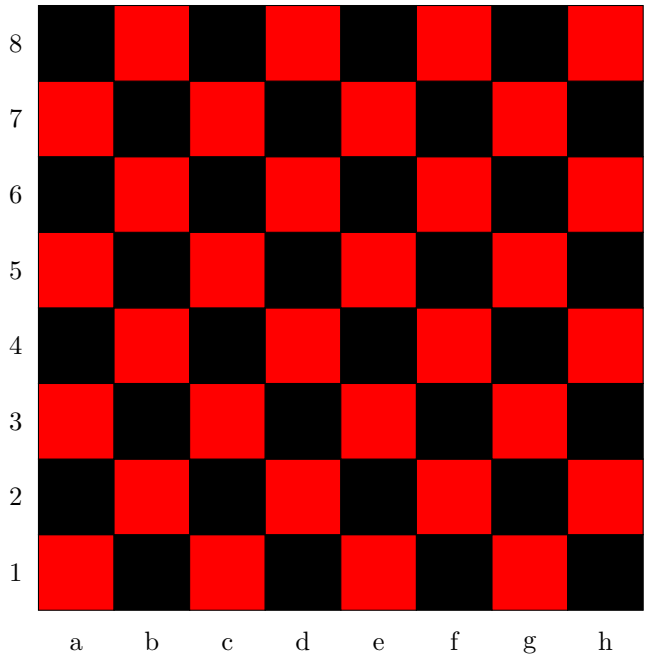

Figure 2: A “flipped” checkerboard

- `'|', '-', '+'`: ignored.
- Whitespace and comments: ignored.

No other input characters are allowed.

A “move” is a sequence of “board coordinates” separated by the string `"->"`. For example: the move `"f2->e3"` could indicate movement of a red pawn; the move `"f2->h4"` could indicate the capture of a piece at position `g3`, and the move `"f2->h4->f6->d8"` could indicate a triple jump if the opponent has pieces at positions `g3`, `g5`, and `e7`. Whitespace is not allowed within a move, and is used to indicate the end of a move.

The following input files are equivalent, and represent the starting game configuration and a few moves.

3.1 Example 1

```
# Start of a game

#
# Much of this file structure will become more important
# in later parts of the project.
#

RULES:
  no capture      # Don't enforce the rule that pieces must be captured
  multiple jumps  # Allow multiple jumps

TURN:
  red

BOARD:
```

```
 " | b | " | b | " | b | " | b
---+---+---+---+---+---+---+---
 b | " | b | " | b | " | b | "
---+---+---+---+---+---+---+---
 " | b | " | b | " | b | " | b
---+---+---+---+---+---+---+---
 . | " | . | " | . | " | . | "
---+---+---+---+---+---+---+---
 " | . | " | . | " | . | " | .
---+---+---+---+---+---+---+---
 r | " | r | " | r | " | r | "
---+---+---+---+---+---+---+---
 " | r | " | r | " | r | " | r
---+---+---+---+---+---+---+---
 r | " | r | " | r | " | r | "
```

```
MOVES:
# Red             black
e3->d4            b6->c5
d4->b6
```

3.2 Example 2

```
RULES:     no capture multiple jumps
TURN:      red
BOARD:
" b " b " b " b
b " b " b " b "
" b " b " b " b
. " . " . " . "
" . " . " . " .
r " r " r " r "
" r " r " r " r
r " r " r " r "
MOVES:     e3->d4 b6->c5 d4->b6
```

3.3 Example 3 (legal but not very readable for humans)

```
RULES:no capture multiple jumps TURN:red BOARD:"b"b"b"bb"b"b"b""b"b"b
"b."."."""."."."."r"r"r"r""r"r"r"rr"r"r"r" MOVES:e3->d4 b6->c5 d4->b6
```

3.4 Output

```
VALID INPUT
Initial configuration:
Turn: red
Red: 0 kings, 12 pawns
Black: 0 kings, 12 pawns
```

4 What you need to check for

What constitutes an "invalid input file"? For this part of the project, you should give errors for the following.

- Formatting errors. For example, if you expect to see a keyword next, and instead get some other text (other than a comment or whitespace), you should give an appropriate error message. Unexpected or extra characters while reading a board configuration is another example of a formatting error.
- Board errors. The board must alternate between red and black squares, so finding a piece where it shouldn't be, or an empty black square where there should be a red one, are examples of board errors. You may also assume that a standard board (as shown in Figure 1) is used (i.e., give an error if the first square is not red). You should **not** check for "too many pieces"; as long as the pieces are in correct positions, you may consider the board configuration to be correct.
- Move errors. You should check that the board coordinates given are valid. For **this part** of the project, you do **not** need to check that the moves are legal moves (e.g., the piece at the starting location can in fact move to the destination).

All error messages should be directed to standard error, and should include the line number of the input file that causes the error. You may stop processing after the first error is encountered.

5 Extra credit (required for groups)

For extra credit, handle input files whose board configuration is using a "flipped board" as shown in Figure 2, in addition to the proper board shown in Figure 1.