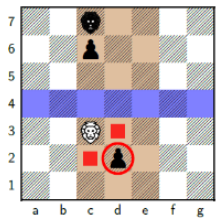
Question: Submission: Setting the board and Generate Lion Moves Writ...

Please help with C++ Code of the following:

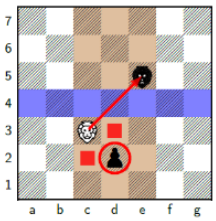
Submission: Setting the board and Generate Lion Moves

Write a C++ program that accepts a FEN string (as described below), stores the piece location information in appropriate data structures, and then outputs the location of all pieces on the board, as well as the side whose move it is to play and then outputs the valid lion moves that can be made by whichever player it is to move. As a reminder, the lion moves as follows:

Lion: The lion moves and captures one square in any direction (including diagonally), but it may not leave its 3x3 castle (highlighted in brown in the diagrams). However, there is one exception to this rule: a lion is allowed to move in a straight or diagonal line across the river if it is able to immediately **capture the opposing lion**. If a player's lion is captured, that player immediately loses the game. The diagram below shows the moves the white lion is able to make.



(a) In the current position, White's lion on c3 can move to the squares marked by a red square or can capture Black's pawn on d2 (red circle).



(b) Compared to the position on the left, the White lion on c3 can now also move across the river to capture the Black lion on e5 and win the game (red arrow)! The White lion could also potentially capture Black's lion anywhere along the c-file.

Input

The first line of input is *N*, the number of FEN strings that must be read in as input. *N* lines follow, with each line consisting of a single FEN string. You may assume that each FEN string is a valid position, and so will contain exactly one white lion and one black lion.

Output

For each FEN string *i*, output the location of all pieces on the board, output the valid lion moves. The moves should be printed in alphabetical order, and should be separated by a single space. Each move should be printed enter text. encoding described in move representations(i.e. as <start square><end square>). If there are no valid moves, nothing should be printed.

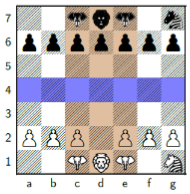
Example Input-Output of Lion Moves :

```
2
2ele1z/ppppppp/7/7/7/PPP1PPP/2ELE1Z w 4
1e1El2/P1P2P1/1P5/7/1E3P1/1P5/4L2 b 79
```

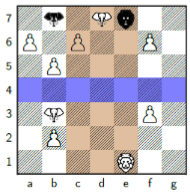
Sample Output

```
d1d2
e7d6 e7d7 e7e1 e7e6
```

Visualisation of Above Test Cases



(a) The White lion can only move to d2, since it is blocked by its other pieces.



(b) The Black lion can move to the empty squares on d6 and e6 and can capture on d7. It cannot move to f7 or f6, since that would be outside of its 3×3 castle. It can also fly across the river to capture the white lion on e1.

Introduction

In this section, we will develop a way of encoding the *STATE* of the game. A single state represents the current board position — it should contain all the information necessary for the game (and successor function) to operate correctly. We will also need to generate the valid moves available to a player given the current state of the game. We will need to write code that accepts a game state as a FEN string and sets up the board. Once this has been done, the next step is to generate the moves available at the current position. This will depend on whether it is white or black to play.

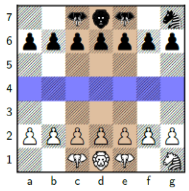
Forsyth-Edwards Notation

Forsyth–Edwards Notation (FEN) is a standard way of encoding chess positions as a string. For the game Congo, the FEN string will consist of three fields, each separated by a blank space:

<position of pieces> <side to move> <move number>

Position of pieces

In our FEN representation, we will specify the placement of each piece on the board. Each piece is encoded as a single character—white pieces uses capital letters, and black pieces use the corresponding lowercase letter. The pieces are: pawn (P), elephant (E), lion (L) and zebra (Z).



The starting position for White and Black

The string describes each rank (row), starting from rank 7 to rank 1. The string for each rank specifies the location of a piece on that rank, and any number of empty squares. Each rank is separated by a /. The easiest way to understand this encoding is to look at some examples, as illustrated below.

Side to move

The side to move is just a single character W or B that indicates whose turn it is to play

Move number

The move number records the number of moves played in the game. It is incremented only after black plays a move, and so a value of *N* indicates that both white and black have made *N* moves.

1. Position of pieces is a string that specifies the placement of each piece on the board. Each rank is described, starting from rank 7 and ending with rank 1.

Move Representations

For every submission, we will represent a move as a string <start_square><end_square> specifying the starting location of the piece to move and then square the piece ends up on. For example, the move e3e4represents a piece moving from e3 to e4.

Input Hint

Hint: a reminder not to be careful when mixing `cin` with `getline`. An example of doing so is below:

```
intN; cin>>N;
cin.ignore(); //NB!
for(inti=0; i<N;++i) { string fen;
    getline(cin, fen);
}
```

Expert Answer

Anonymous answered this
52 answers

Source code for main.cpp:

```
#include <iostream>
#include <vector>

using namespace std;

class Node{
public:
char file;
int rank;
char value;
string position;

Node(int column, int row, char v){
file = char(column+96);
rank = row;
position = file + to_string(rank);
value = v;
}
};

class Grid{
public:
vector<Node> grid;
string positionOfPieces;
string sideToMove;
int moveNumber;

vector<vector<string>> peicesPositions = {{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{}};

//constructor
Grid(string position, char side, int moveNum){
positionOfPieces = position;
if(side == 'w'){
sideToMove = "white";
}
else{
sideToMove = "black";
}
moveNumber = moveNum;
}

void createNode(int x, int y, char v){
Node node(x, y, v);
grid.push_back(node);
}

void createGrid(){
int i = 7;
int j = 1;
for(int position = 0; position < positionOfPieces.length(); position++){
char c = positionOfPieces[position];
if(isdigit(c)){
int intC = int(c)-48;
for(int k = 0; k < intC; k++){
createNode(j, i, 'y');
j++;
}
}
else if(c == '/'){
i--;
j = 1;
}
else{
createNode(j, i, c);
j++;
}
}
}

void printGrid(){
for(int i = 0; i < 7; i++){
for(int j = 0; j < 7; j++){
if(j != 6){
cout << grid[i * 7 + j].value << " ";
}
else{
cout << grid[i * 7 + j].value << endl;
}
}
```

```
Node getNode(int file, int rank){
if(rank == 0 || file == 0){
cout << "Rank or File should not be 0!\n";
}
return grid[49 - 7*rank + file -1];
}

void printNodeInfo(int file, int rank){
Node node = getNode(file, rank);
cout << node.position << " = " << node.value << endl;
}

void addLocationOfPieces(){
for(int i = 1; i <= 7; i++){
for(int j = 1; j <= 7; j++){
Node node = getNode(i, j);
char peice = node.value;
if(peice == 'P'){
peicesPositions[0].push_back(node.position);
}
else if(peice == 'p'){
peicesPositions[1].push_back(node.position);
}
else if(peice == 'S'){
peicesPositions[2].push_back(node.position);
}
else if(peice == 's'){
peicesPositions[3].push_back(node.position);
}
else if(peice == 'G'){
peicesPositions[4].push_back(node.position);
}
else if(peice == 'g'){
peicesPositions[5].push_back(node.position);
}
else if(peice == 'M'){
peicesPositions[6].push_back(node.position);
}
else if(peice == 'm'){
peicesPositions[7].push_back(node.position);
}
else if(peice == 'E'){
peicesPositions[8].push_back(node.position);
}
else if(peice == 'e'){
peicesPositions[9].push_back(node.position);
}
else if(peice == 'L'){
peicesPositions[10].push_back(node.position);
}
else if(peice == 'l'){
peicesPositions[11].push_back(node.position);
}
else if(peice == 'C'){
peicesPositions[12].push_back(node.position);
}
else if(peice == 'c'){
peicesPositions[13].push_back(node.position);
}
else if(peice == 'Z'){
peicesPositions[14].push_back(node.position);
}
else if(peice == 'z'){
peicesPositions[15].push_back(node.position);
}
}
}
}

void printLocationOfPieces(){
for(int i = 0; i < 16; i++){
if(i == 0){
cout << "white pawn:";
}
else if(i == 1){
cout << "black pawn:";
}
else if(i == 2){
cout << "white superpawn:";
}
else if(i == 3){
cout << "black superpawn:";
}
else if(i == 4){
cout << "white giraffe:";
```

```
}
else if(i == 6){
cout << "white monkey:";
}
else if(i == 7){
cout << "black monkey:";
}
else if(i == 8){
cout << "white elephant:";
}
else if(i == 9){
cout << "black elephant:";
}
else if(i == 10){
cout << "white lion:";
}
else if(i == 11){
cout << "black lion:";
}
else if(i == 12){
cout << "white crocodile:";
}
else if(i == 13){
cout << "black crocodile:";
}
else if(i == 14){
cout << "white zebra:";
}
else if(i == 15){
cout << "black zebra:";
}
int size = peicesPositions[i].size();
for(int j = 0; j < size; j++){
cout << " " << peicesPositions[i][j];
}
cout << endl;
}
}

void printSideToMove(){
cout << "side to play: " << sideToMove;
}
};

int main(){

//read FEN string
int N;
cin >> N;
vector<string> positionOfPiecesArray(N);
vector<char> sideToMoveArray(N);
vector<int> moveNumberArray(N);
string position;
char side;
int moveNum;
for(int i = 0; i < N; i++){
cin >> position >> side >> moveNum;
positionOfPiecesArray[i] = position;
sideToMoveArray[i] = side;
moveNumberArray[i] = moveNum;
}

//create a grid
for(int i = 0; i < N; i++){
Grid grid(positionOfPiecesArray[i], sideToMoveArray[i], moveNumberArray[i]);
grid.createGrid();
// grid.printGrid();
grid.addLocationOfPieces();
grid.printLocationOfPieces();
grid.printSideToMove();
if(i != N-1){
cout << endl << endl;
}
}

return 0;
}
```

Output screenshot:

```
1m1E12/P1P2P1/1S4C/4S2/1E3S1/1P3c1/2GL3 b 79
white pawn: a2 a6 b2 b6 c2 c6 d2 d6 e2 e6 f2 f6 g2 g6
black pawn:
white superpawn:
black superpawn:
white giraffe: a1
black giraffe: a7
white monkey: b1
black monkey: b7
white elephant: c1 e1
black elephant: c7 e7
white lion: d1
black lion: d7
white crocodile: f1
black crocodile: f7
white zebra: g1
black zebra: g7
side to play: white

white pawn: a6 b2 c6 f6
black pawn:
white superpawn: b5 e4 f3
black superpawn:
white giraffe: c1
black giraffe:
white monkey:
black monkey: b7
white elephant: b3 d7
black elephant:
white lion: d1
black lion:
white crocodile: g5
black crocodile: f2
white zebra:
black zebra:
side to play: black

...Program finished with exit code 0
Press ENTER to exit console.
```

Hide comments (1)

Comments

Anonymous posted 4 days ago

Can you please help : Write a C++ program that accepts a FEN string, stores the piece location information in appropriate data structures, and then outputs the valid lion moves that can be made by whichever player it is to move. As required in the question.

Leave a comment