Question: 2 Forsyth-Edwards Notation Forsyth-Edwards Notation (FEN) i...

Please assist with the following , please also read the requirements carefully

2 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:

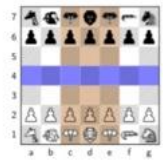
```
<position of pieces> <side to move> <move number>
```

2.1 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), giraffe (G), monkey (M), elephant (E), lion (L), crocodile (C), zebra (Z) and superpawn (S).

The string describes each rank (row), starting from rank 7 to rank 1. The string for each rank specifices 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.

1



(a) The starting position with the representation gmelecz/ppppppp/7/7/7/PPPPPPP/GMELECZ. This string indicates that rank 7 consists of black pieces only and that they are: giraffe, monkey, elephant, lion, elephant, crocodile, zebra. The next rank is 6, which contains all black pawns. The next rank is 5 which has the representation 7. This indicates that there are 7 empty squares. The same is true for ranks 4–3. Rank 2 consists of only white pawns, while rank 1 (GMELECZ) indicates white giraffe, monkey, elephant, lion, elephant, crocodile, zebra.

(b) The above board is represented by 214/7/4z2/4c2/PP2EP1/3L2p/7. The 7th rank (214) has two empty squares, a black lion, and 4 empty squares. The 6th rank has 7 empty squares. The 5th rank (4z2) has 4 empty squares, a black zebra and 2 empty squares. The 4th rank (4c2) has 4 empty squares, a black crocodile and 2 empty squares. The 3rd rank (PP2EP1) has two white pawns, then two empty spaces, a white elephant, a white pawn and an empty space. The 2nd rank (3L2p) has 3 empty squares, a white lion, two empty squares and a black pawn. Finally, rank 1 has 7 empty squares.

2.2 Side to move

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

2.3 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.

on, you may assume that the game starts with none of these pieces on the board. If a pawn reaches the end rank, it does not get promoted to a superpawn. The new starting position will be

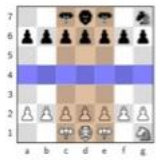


Figure 1: The starting position for White and Black excluding giraffes, monkeys and crocodiles.

1 Introduction

In this section, we will 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 (see Part 1 if you have no already done so). 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.

1

2 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 e3e4 represents a piece moving from e3 to e4.

3 Input Hint

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

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

Generate Zebra Moves

Write a C++ program that accepts a FEN string, stores the piece location information in appropriate data structures, and then outputs the valid zebra moves that can be made by whichever player it is to move. As a reminder, the zebra moves two squares vertically and one square horizontally, or two squares horizontally and one square vertically (with both forming the shape of an L). The zebra can jump over pieces to reach its destination.

(a) The White zebra on e2 can move to any square indicated by the arrows. It cannot move to c1, since that square is occupied by another White piece. Note that the pawn on f3 does not prevent the zebra from jumping to f4 or g3.

(b) The Black zebra can move to any square indicated by the arrows and can thus choose to capture the White piece on f5.

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. The FEN string may contain 0 or 1 Black and White zebras.

Output

For each FEN string i, output the valid zebra moves. The moves should be printed in alphabetical order, and should be separated by a single space. Each move should be printed using the encoding described in Section 2. If there are no valid moves, or no zebras for the side to move exist, nothing should be printed.

5

Example Input-Output

Sample Input

```
2
6E/3p11p/ez24/E6/2L4/3p3/7 w 45
7/7/21P1p1/7/2ppeE1/1z4P/4L2 b 32
```

Sample Output

```
c5a6 c5b3 c5b7 c5d3 c5d7 c5e4 c5e6
b2a4 b2c4 b2d1
```

Show transcribed image text

Expert Answer

Anonymous answered this

```
#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";
```

```
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 == 5){
cout << "black giraffe:";
}
else if(i == 6){
```

```
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.
```

Comment >