

**Chess game**



**Group Members**

**Muhammad Bin Ahsan - 076**

**Shaariff Mujtaba Bhatti – 112**

**Laiba Tahir-057**

**CODE:**

#include <iostream>

#include <string>

#include <cctype>

#include <cmath>

using namespace std;

const int BOARD\_SIZE = 8;

char board[BOARD\_SIZE][BOARD\_SIZE];

// Initialize the chessboard

void initializeBoard() {

for (int i = 0; i < BOARD\_SIZE; ++i) {

board[1][i] = 'p'; // Black pawns

board[6][i] = 'P'; // White pawns

}

const char white\_pieces[] = { 'R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R' };

const char black\_pieces[] = { 'r', 'n', 'b', 'q', 'k', 'b', 'n', 'r' };

for (int i = 0; i < BOARD\_SIZE; ++i) {

board[0][i] = black\_pieces[i];

board[7][i] = white\_pieces[i];

}

for (int i = 2; i < 6; ++i) {

for (int j = 0; j < BOARD\_SIZE; ++j) {

board[i][j] = '-';

}

}

}

// Print the chessboard

void printBoard() {

cout << " a b c d e f g h\n";

for (int i = BOARD\_SIZE - 1; i >= 0; --i) {

cout << i + 1 << " ";

for (int j = 0; j < BOARD\_SIZE; ++j) {

cout << board[i][j] << " ";

}

cout << i + 1 << "\n";

}

cout << " a b c d e f g h\n";

}

// Convert chess notation to board coordinates

bool parsePosition(const string& position, int& row, int& col) {

if (position.length() != 2) return false;

col = position[0] - 'a';

row = position[1] - '1';

return row >= 0 && row < BOARD\_SIZE && col >= 0 && col < BOARD\_SIZE;

}

// Validate pawn moves

bool isValidPawnMove(int srcRow, int srcCol, int destRow, int destCol, char player) {

int direction = (player == 'w') ? -1 : 1; // White pawns move up, black down

bool isOpponentPiece = (player == 'w') ? islower(board[destRow][destCol]) : isupper(board[destRow][destCol]);

// Regular move

if (srcCol == destCol && board[destRow][destCol] == '-') {

return (destRow - srcRow == direction) ||

(srcRow == (player == 'w' ? 6 : 1) && destRow - srcRow == 2 \* direction);

}

// Capture move

if (abs(srcCol - destCol) == 1 && destRow - srcRow == direction && isOpponentPiece) {

return true;

}

return false;

}

// Validate rook moves

bool isValidRookMove(int srcRow, int srcCol, int destRow, int destCol, char player) {

if (srcRow != destRow && srcCol != destCol) return false;

int step = (srcRow == destRow) ? (srcCol < destCol ? 1 : -1) : (srcRow < destRow ? 1 : -1);

for (int i = srcRow + (srcRow != destRow ? step : 0), j = srcCol + (srcCol == destCol ? step : 0);

(srcRow == destRow ? j != destCol : i != destRow);

i += (srcRow != destRow ? step : 0), j += (srcCol == destCol ? step : 0)) {

if (board[i][j] != '-') return false;

}

return board[destRow][destCol] == '-' || islower(board[destRow][destCol]) != (player == 'w');

}

// Validate bishop moves

bool isValidBishopMove(int srcRow, int srcCol, int destRow, int destCol, char player) {

if (abs(srcRow - destRow) != abs(srcCol - destCol)) return false;

int rowStep = (srcRow < destRow) ? 1 : -1;

int colStep = (srcCol < destCol) ? 1 : -1;

for (int r = srcRow + rowStep, c = srcCol + colStep; r != destRow; r += rowStep, c += colStep) {

if (board[r][c] != '-') return false;

}

return board[destRow][destCol] == '-' || islower(board[destRow][destCol]) != (player == 'w');

}

// Validate knight moves

bool isValidKnightMove(int srcRow, int srcCol, int destRow, int destCol, char player) {

int rowDiff = abs(srcRow - destRow);

int colDiff = abs(srcCol - destCol);

char destPiece = board[destRow][destCol];

return (rowDiff == 2 && colDiff == 1 || rowDiff == 1 && colDiff == 2) &&

(destPiece == '-' || islower(destPiece) != (player == 'w'));

}

// Validate queen moves

bool isValidQueenMove(int srcRow, int srcCol, int destRow, int destCol, char player) {

return isValidRookMove(srcRow, srcCol, destRow, destCol, player) ||

isValidBishopMove(srcRow, srcCol, destRow, destCol, player);

}

// Validate king moves

bool isValidKingMove(int srcRow, int srcCol, int destRow, int destCol, char player) {

int rowDiff = abs(srcRow - destRow);

int colDiff = abs(srcCol - destCol);

char destPiece = board[destRow][destCol];

return rowDiff <= 1 && colDiff <= 1 && (destPiece == '-' || islower(destPiece) != (player == 'w'));

}

// Validate moves for a piece

bool isValidMove(int srcRow, int srcCol, int destRow, int destCol, char player) {

char piece = board[srcRow][srcCol];

if (player == 'w' && !isupper(piece)) return false;

if (player == 'b' && !islower(piece)) return false;

piece = tolower(piece);

switch (piece) {

case 'p': return isValidPawnMove(srcRow, srcCol, destRow, destCol, player);

case 'r': return isValidRookMove(srcRow, srcCol, destRow, destCol, player);

case 'n': return isValidKnightMove(srcRow, srcCol, destRow, destCol, player);

case 'b': return isValidBishopMove(srcRow, srcCol, destRow, destCol, player);

case 'q': return isValidQueenMove(srcRow, srcCol, destRow, destCol, player);

case 'k': return isValidKingMove(srcRow, srcCol, destRow, destCol, player);

default: return false;

}

}

// Execute a move

bool makeMove(const string& move, char player) {

int srcRow, srcCol, destRow, destCol;

string src = move.substr(0, 2), dest = move.substr(3, 2);

if (!parsePosition(src, srcRow, srcCol) || !parsePosition(dest, destRow, destCol)) {

cout << "Invalid move format!\n";

return false;

}

if (isValidMove(srcRow, srcCol, destRow, destCol, player)) {

board[destRow][destCol] = board[srcRow][srcCol];

board[srcRow][srcCol] = '-';

return true;

}

cout << "Invalid move!\n";

return false;

}

// Main game loop

void playGame() {

char currentPlayer = 'w';

string move;

while (true) {

printBoard();

cout << (currentPlayer == 'w' ? "White" : "Black") << "'s turn. Enter move (e.g., e2 e4): ";

getline(cin, move);

if (move == "quit") {

cout << "Game ended!\n";

break;

}

if (makeMove(move, currentPlayer)) {

currentPlayer = (currentPlayer == 'w' ? 'b' : 'w');

}

}

}

int main() {

initializeBoard();

playGame();

return 0;

}