#include<bits/stdc++.h>

using namespace std;

char board[3][3] =

{

{ 'x', 'o', 'x' },

{ 'o', 'o', 'x' },

{ '\_', '\_', '\_' }

};

int minimax(char board[3][3], int depth, bool isMax);

void printBoard()

{

cout<<"socre:"<<minimax(board,0,false);

for(int i=0;i<3;i++)

cout<<"\n"<<board[i][0]<<" |"<<board[i][1]<<" |"<<board[i][2]<<" |";

cout<<"\n\n";

}

struct Move

{

int row, col;

};

char player = 'x', opponent = 'o';

bool isMovesLeft(char board[3][3])

{

for (int i = 0; i<3; i++)

for (int j = 0; j<3; j++)

if (board[i][j]=='\_')

return true;

return false;

}

int evaluate(char b[3][3])

{

// Checking for Rows for X or O victory.

for (int row = 0; row<3; row++)

{

if (b[row][0]==b[row][1] &&

b[row][1]==b[row][2])

{

if (b[row][0]==player)

return +10;

else if (b[row][0]==opponent)

return -10;

}

}

for (int col = 0; col<3; col++)

{

if (b[0][col]==b[1][col] &&

b[1][col]==b[2][col])

{

if (b[0][col]==player)

return +10;

else if (b[0][col]==opponent)

return -10;

}

}

if (b[0][0]==b[1][1] && b[1][1]==b[2][2])

{

if (b[0][0]==player)

return +10;

else if (b[0][0]==opponent)

return -10;

}

if (b[0][2]==b[1][1] && b[1][1]==b[2][0])

{

if (b[0][2]==player)

return +10;

else if (b[0][2]==opponent)

return -10;

}

return 0;

}

int minimax(char board[3][3], int depth, bool isMax)

{

int score = evaluate(board);

// If Maximizer has won the game return his/her

// evaluated score

if (score == 10)

return score;

// If Minimizer has won the game return his/her

// evaluated score

if (score == -10)

return score;

// If there are no more moves and no winner the

// it is a tie

if (isMovesLeft(board)==false)

return 0;

// If this maximizer's move

if (isMax)

{

int best = -1000;

// Traverse all cells

for (int i = 0; i<3; i++)

{

for (int j = 0; j<3; j++)

{ // Check if cell is empty

if (board[i][j]=='\_')

{

// Make the move

board[i][j] = player;

// Call minimax recursively and choose

// the maximum value

best = max( best,

minimax(board, depth+1, !isMax) );

printBoard();

// Undo the move

board[i][j] = '\_';

}

}

}

return best;

}

// If this minimizer's move

else

{

int best = 1000;

// Traverse all cells

for (int i = 0; i<3; i++)

{

for (int j = 0; j<3; j++)

{

// Check if cell is empty

if (board[i][j]=='\_')

{

// Make the move

board[i][j] = opponent;

// Call minimax recursively and choose

// the minimum value

best = min(best,

minimax(board, depth+1, !isMax));

printBoard();

// Undo the move

board[i][j] = '\_';

}

}

}

return best;

}

}

// This will return the best possible move for the player

Move findBestMove(char board[3][3])

{

int bestVal = -1000;

Move bestMove;

bestMove.row = -1;

bestMove.col = -1;

// Traverse all cells, evaluate minimax function for

// all empty cells. And return the cell with optimal

// value.

for (int i = 0; i<3; i++)

{

for (int j = 0; j<3; j++)

{

// Check if cell is empty

if (board[i][j]=='\_')

{

// Make the move

board[i][j] = player;

// compute evaluation function for this

// move.

int moveVal = minimax(board, 0, false);

// Undo the move

board[i][j] = '\_';

// If the value of the current move is

// more than the best value, then update

// best/

if (moveVal > bestVal)

{

bestMove.row = i;

bestMove.col = j;

bestVal = moveVal;

}

}

}

}

printf("The value of the best Move is : %d\n\n",

bestVal);

if(bestVal>0)

printf("X will win the game\n");

else if(bestVal<0)

printf("O will win the game\n");

else

printf("game will draw\n");

return bestMove;

}

//Main Function

int main()

{

Move bestMove = findBestMove(board);

return 0;

}