#include <iostream>

#include <cstring>

#include <algorithm>

#include <ctime>

using namespace std;

#define compareBoxes(box1, box2, box3) ((board[box1] == board[box2]) && (board[box2] == board[box3]) && (board[box1] != 0)) //Checkes if three items are the same, and makes sure they're not 0's.

#define numberToLetter(x) ((x > 0) ? (x == 1) ? 'X' : 'O' : ' ') //Takes the number and turns it into the letter or space.

int getWinner(int board[9]) {

//Finds winner of game, if there is no winner, returns 0.

int winner = 0;

for (int x = 0; x < 3; x++) {

if (compareBoxes(3\*x, 3\*x+1, 3\*x+2)) { //Chekcs rows.

winner = board[3\*x];

break;

} else if (compareBoxes(x, x+3, x+6)) { //Checks columns.

winner = board[x];

break;

} else if (compareBoxes(2\*x, 4, 8-2\*x) && (x < 2)) { //Checks diagonals. Doesn't check if x == 2.

winner = board[4];

break;

}

}

return winner;

}

bool gameOver(int board[9]){

//Checks if game is over, and announces who won, or if it was a tie.

int winner = getWinner(board);

if (winner > 0) {

cout << numberToLetter(winner) << " wins!"<< endl;

return true;

}

for (int x = 0; x < 9; x++) {

if (board[x] == 0) return false;

}

cout << "Tie!\n\n";

return true;

}

int willWin(int board[9], int player) {

//Checks if a given player could win in the next plank.

for (int x = 0; x < 9; x++) {

int tempBoard[9];

memcpy(tempBoard, board, 36);

if (board[x] > 0) continue;

tempBoard[x] = player;

if(getWinner(tempBoard) == player) return x;

}

return -1;

}

int exceptionalCase(int board[9]) {

//Finds bords that are exceptions to how the algorithm works.

int cases[2][9] = {{1,0,0,0,2,0,0,0,1}, {0,1,0,1,2,0,0,0,0}}; //Boards that don't work with algorithm.

int answers[2][4] = {{3,3,3,3}, {2,8,6,0}};

int rotatedBoard[9] = {6,3,0,7,4,1,8,5,2};

int newBoard[9];

int tempBoard[9];

for(int x = 0; x < 9; x++) {

newBoard[x] = board[x];

}

for (int caseIndex = 0; caseIndex < 2; caseIndex++) {

for(int rotation = 0; rotation < 4; rotation++) {

for (int x = 0; x < 9; x++)

tempBoard[x] = newBoard[x];

int match = 0;

//Rotates board so it works with different versions of the same board.

for (int box = 0; box < 9; box++) {

newBoard[box] = tempBoard[rotatedBoard[box]];

}

for (int x = 0; x < 9; x++) {

if (newBoard[x] == cases[caseIndex][x]) match++;

else break;

}

if (match == 9) return answers[caseIndex][rotation];

}

}

return -1;

}

int getSpace(int board[9], int spaces[4]) {

//Gets a random corner or side that's not taken.

bool isSpaceEmpty = false;

int y;

for (int x = 0; x < 4; x++) {

if (board[spaces[x]] == 0) {

isSpaceEmpty = true;

break;

}

}

if (isSpaceEmpty) {

do {

y = rand() % 4;

} while (board[spaces[y]] != 0);

return spaces[y];

}

return -1;

}

void outputBoard(int board[9]) {

for(int line = 0; line < 3; line++){

for (int box = 0; box < 3; box++) {

cout << numberToLetter(board[3\*line+box]) << ((box < 2) ? '|' : '\n');

}

cout << ((line < 2) ? "-----\n" : "\n");

}

}

int main(){

int board[9] = {0,0,0,0,0,0,0,0,0}; //Starts empty board.

int possibleWinner;

int move;

bool isInvalid;

string moveString;

srand((int) time(0));

int corners[4] = {0,2,6,8};

int sides[4] = {1,3,5,7};

cout << "1|2|3\n-----\n4|5|6\n-----\n7|8|9\n\n";

while (true) {

//Player X decides what move they'll do.

do {

cout << "X: ";

getline(cin, moveString);

move = moveString[0] - '1';

if (move > 8 || move < 0 || board[move] != 0) {

cout << "Invalid input" << endl;

isInvalid = true;

} else {

board[move] = 1;

isInvalid = false;

cout << endl;

}

} while (isInvalid);

//Decides whether or not the game continues.

if (gameOver(board) > 0) {

outputBoard(board);

break;

}

//Player O decides which move they'll do.

bool good = false;

for (int x = 2; x > 0; x--){

possibleWinner = willWin(board, x);

if (possibleWinner != -1) {

board[possibleWinner] = 2;

good = true;

break;

}

}

if (good);

else if (board[4] == 0) board[4] = 2; //Middle.

else if (exceptionalCase(board) > -1) board[exceptionalCase(board)] = 2; //Exception boards.

else if (getSpace(board, corners) != -1) board[getSpace(board, corners)] = 2; //Corners

else board[getSpace(board, sides)] = 2; //Sides

//Prints the board to the screen.

outputBoard(board);

//Decides whether or not the game continues.

if(gameOver(board)) break;

}

return 0;

}