**W4 -** HOMEWORK

*2D Array Problem Solving*

## *At the end of this practice, you should be able to…*

* Practice **array manipulations** with functions
* Consolidate **Top-Down Design** and function-based refactoring applied to 2D arrays
* Apply **scope**, **memory**, and **function call** concepts

## *How do we structure exercises?*

We organize each practice into 3 parts:

| ANALYSE | **Understand** existing codes, find the **bugs** or **complete** missing gaps |
| --- | --- |
| MANIPULATE | Ensure you can **apply the theory** with some basic challenges |
| CREATE | **Express your creativity** with more complex challenges |

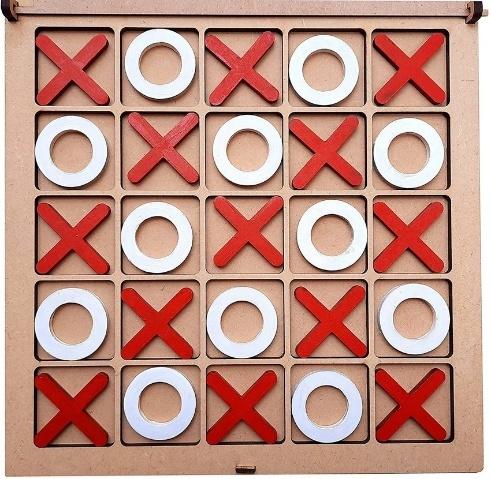
## *Are you lost?*

You can read the following documentation to be ready for this practice

<https://www.w3schools.com/c/c_arrays.php>

**CREATE**

**PROBLEM (The Tic Tac Toe Game)**

****

In this exercise, you will implement a simple 2-player Tic-Tac-Toe game in C, played on the console. This exercise will help you to:

* Manipulate **2D arrays**.
* Strengthen your understanding of **modular code** and **problem decomposition**.

** Using chat GPT is cheating. You might get a 0 to this homework or event for the whole term.**

**Rules**

* The game is played on a 5×5 grid.
* Two players take turns to place their symbol (Player 1 = 'X' and Player 2 = 'O').
* The first player to align **4 of their symbols** in a row (*horizontally, vertically, or diagonally*) wins.
* If the board is **filled** and **no one has won**, the game ends in a **draw**.

💡 *Note: You need to check for 4-in-a-row, not 5.*

**Examples**

On the bellow grid, the player 1 has won, with 4 symbols on the first column:

{ 'X', 'O', 'O', 'X', 'X' },

{ 'X', 'X', 'O', 'O', 'X' },

{ 'X', 'X', 'O', 'X', 'O' },

{ 'X', 'O', 'X', 'O', 'X' },

{ 'O', 'O', 'O', 'X', 'X' },

**Game Flow**

* Start by initializing the game board. We start with empty cells (e.g., ' ').
* Display the game board on the console
* *As long as no winner or no raw game :*
  + Each player (player 1, player 2) enters a position (row, com)
  + The position shall refer to an empty cell, and shall be inside the board game
  + **Check for a win** (*4 same symbols in a row - horizontal, vertical, or diagonal)*
  + **Check for a draw** (*the board is full and there is no winner*)
  + Alternate turns
* Display the game result (player 1 won, player 2 won, draw game)

**Scenario Example**

Player 1 turn.

Enter row (0-4): 2

Enter column (0-4): 1

Current Board:

. . . . .

. . . . .

. X . . .

. . . . .

. . . . .

Player 2 turn.

Enter row (0-4): 0

Enter column (0-4): 0

Current Board:

O . . . .

. . . . .

. X . . .

. . . . .

. . . . .

(Player 1 and 2 continue making moves…)

*After a few more turns:*

O . . . .

. O . . .

. X X X X

. . . O .

. . . . .

🎉 Player X wins with a horizontal line of 4!

**Top-Down Design**

* You actually need to divide your whole game into small tasks (functions)
* You can decide either to define your game board globally or locally to the main and pass it to your functions

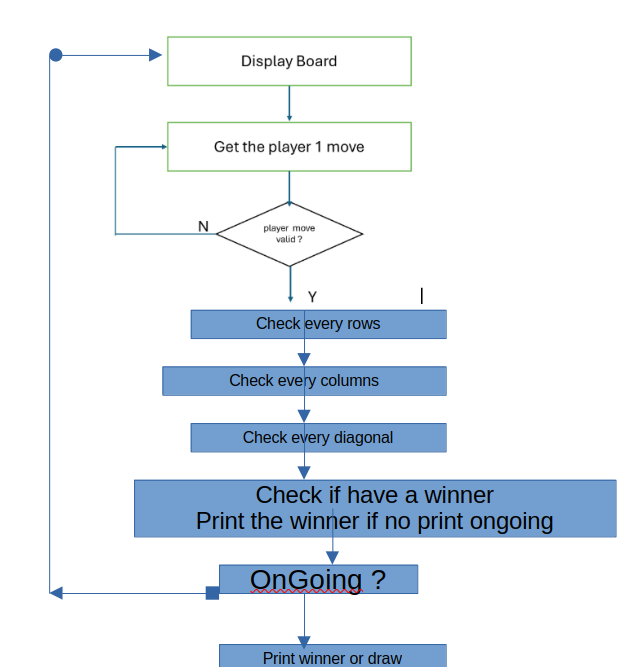
**Q1 –** Complete the bellow tables with your function definitions

*You can find below a few examples of functions:*

| **Function** | **Parameters** | **Return** | **Description** |
| --- | --- | --- | --- |
|  |  |  |  |
| makeMove | int row  int column  bool isPlayer1 | void | Make a move (for player 1 or player 2) |
| displayBoard |  | void | Display the board on console |
| checkRow | char board  char name | bool rowq | check to see if every rows have a winner |
| checkColumn | char board  char name | bool columnq | check to see if every columns have a winner |
| checkDiagonal | char board  char name | bool diagonal | check to see if both diagonals have a winner |
| onGoing | char board | bool ongiong | check to see if the game is still on going , win or draw |
| gameLogic | char board  int draw | bool onGoingX | to print who wins or ongoing |

**Q2 –** Draw **your game flowchart**

*You can find below an example of flow:*



**Q3 – Code the game logic – Without console interaction**

Start with the game logic first : given a game board, identify the winner or draw game.

*Start code*

#include <stdbool.h>

#include <stdio.h>

#define GAME\_SIZE 5 // Game row and columns

#define WIN\_COUNT 4 // Game row and columns

#define PLAYER\_1\_SYMBOL 'X'

#define PLAYER\_2\_SYMBOL 'O'

#define PLAYER\_1\_WON 1

#define PLAYER\_2\_WON 2

#define DRAW\_GAME 3

// -------------------------------------------------------------------

// GLOBAL VARIABLES

// -------------------------------------------------------------------

char board[5][5] = {

{'.', '.', '.', '.', '.'}, // Row 1

{'O', 'O', '.', '.', '.'}, // Row 2

{'X', 'X', 'X', 'X', '.'}, // Row 3 ← Winning rowz

{'.', '.', 'O', '.', '.'}, // Row 4

{'.', '.', '.', '.', '.'} // Row 5

};

// -------------------------------------------------------------------

// YOUR FUNCTIONS

// -------------------------------------------------------------------

// -------------------------------------------------------------------

// MAIN

// -------------------------------------------------------------------

int main() {

    printf("%d", checkGame()); // Check player 1 won

    return 0;

}

*Example board version to check the draw game*

char board[5][5] = {

{'X', 'O', 'X', 'O', 'X'},

{'O', 'X', 'O', 'X', 'O'},

{'X', 'O', 'X', 'O', 'X'},

{'O', 'X', 'O', 'X', 'O'},

{'O', 'X', 'O', 'X', 'O'}

};

**Q4 –** Code the console interaction  **– BONUS**

As a bonus, you can code the console interaction:

* By asking for each round the players’ positions and validate them
* By updating the game board
* By check if any game end and displaying the game changes on console

#include <stdio.h>

#include <stdbool.h>

#define WIN\_COUNT 4 // Game row and columns

#define GRID\_SIZE 5

#define PLAYER1 'X'

#define PLAYER2 'O'

// Global variable

char board[GRID\_SIZE][GRID\_SIZE] = {

{'.', '.', '.', '.', '.'}, // Row 1

{'.', '.', '.', '.', '.'}, // Row 2

{'.', '.', '.', '.', '.'}, // Row 3

{'.', '.', '.', '.', '.'}, // Row 4

{'.', '.', '.', '.', '.'} // Row 5

};

// Function prototype

bool checkRow(char board[GRID\_SIZE][GRID\_SIZE], char name){

for (int row = 0; row < GRID\_SIZE; row++){

int found = 0;

for (int column = 0; column < GRID\_SIZE; column++){

if (board[row][column] == name ) {

found++;

if(found == WIN\_COUNT) return true;

}else found = 0;

}

}

return false;

}

bool checkColumn(char board[GRID\_SIZE][GRID\_SIZE], char name){

for (int column = 0; column < GRID\_SIZE; column++){

int found = 0;

for (int row = 0; row < GRID\_SIZE - 1; row++){

if (board[row][column] == name && board[row + 1][column]) {

found++;

if(found == WIN\_COUNT) return true;

}else found = 0;

}

}

return false;

}

bool checkDiagonal(char board[GRID\_SIZE][GRID\_SIZE], char name){

// Top-left to bottom-right

for (int i = 0; i <= GRID\_SIZE - WIN\_COUNT; i++) {

for (int j = 0; j <= GRID\_SIZE - WIN\_COUNT; j++) {

int count = 0;

for (int k = 0; k < WIN\_COUNT; k++) {

if (board[i + k][j + k] == name) count++;

else break;

}

if (count == WIN\_COUNT) return true;

}

}

// Top-right to bottom-left

for (int i = 0; i <= GRID\_SIZE - WIN\_COUNT; i++) {

for (int j = WIN\_COUNT - 1; j < GRID\_SIZE; j++) {

int count = 0;

for (int k = 0; k < WIN\_COUNT; k++) {

if (board[i + k][j - k] == name) count++;

else break;

}

if (count == WIN\_COUNT) return true;

}

}

return false;

}

bool onGoing(char board[GRID\_SIZE][GRID\_SIZE]){

for (int row = 0; row < GRID\_SIZE ; row++){

for (int column = 0; column < GRID\_SIZE ; column++){

if (board[row][column] == '.') return true;

}

}

return false;

}

void gameInput(char board[GRID\_SIZE][GRID\_SIZE], int player){

int row, column;

bool valid = true;

for (; valid;){

bool validRow = true, validColumn = true;

for (; validRow;){

printf("Enter row(0-4) : ");

scanf(" %d", &row);

if (row < GRID\_SIZE) validRow = false;

else printf("Invalid input\n");

}

for (; validColumn;){

printf("Enter column(0-4) : ");

scanf(" %d", &column);

if (column < GRID\_SIZE) validColumn = false;

else printf("Invalid input\n");

}

if (board[row][column] != '.') printf("Input again! This spot is already have an input\n");

else valid = false;

}

if (player == 1) board[row][column] = PLAYER1;

else board[row][column] = PLAYER2;

printf("Current board :\n");

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

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

printf("%c ", board[i][j]);

}

printf("\n");

}

}

bool gameLogic(char board[GRID\_SIZE][GRID\_SIZE]){

bool rowX = checkRow(board, 'X'), rowO = checkRow(board, 'O');

bool columnX = checkColumn(board, 'X'), columnO = checkColumn(board, 'O');

bool diagonalX = checkDiagonal(board, 'X'), diagonalO = checkDiagonal(board, 'O');

bool onGoingX = onGoing(board);

if (rowX || columnX || diagonalX){

printf("Player 1 wins!!!");

onGoingX = false;

}else if (rowO || columnO || diagonalO){

printf("Player 2 wins!!!");

onGoingX = false;

}else if (!onGoingX){

printf("Draw");

}else{

printf("Ongoing\n");

onGoingX = true;

}

return onGoingX;

}

int main(){

bool notGameOver = true;

for (; notGameOver;){

printf("Player 1's turn \n");

gameInput(board, 1);

notGameOver = gameLogic(board);

if (notGameOver){

printf("Player 2's turn : \n");

gameInput(board, 2);

notGameOver = gameLogic(board);

}

}

return 0;

}