REVA UNIVERSITY
Bengaluru, India

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

B.Tech in Computer Science and Engineering

Lab Assignment -01

Name of Course	Data Structures and Algorithms
Course Code	
Year and Semester	3rd year II Semester
Academic Year	2023-2024

Submitted By

Name of the Student					
Section					
Name of the Faculty					
Max. Marks		**Marks Obtained**		**Signature of the Student**	**Signature of the Faculty**

Problem Statement : In this problem, you have to read a field of tic tac toe and find out who is the winner. In the input you have 3 lines with 3 characters in each (representing the tic tac toe field) where 'X' is one player, 'O' is the other player and '_' is a blank field. Output "X" if X is the winner, "O" if O is the winner and "NO" if niether X nor O have won.

INFERENCE :

checkWinner Function:

This function takes a 3x3 character array (box) representing the Tic Tac Toe board as input and checks for a winner. It does this by examining the rows, columns, and diagonals of the board.

- **Checking Rows and Columns:**
  - It uses nested loops to iterate over the rows and columns.
  - For rows, it checks if all elements in a row are equal and not equal to the placeholder character '_' (indicating an empty space).
  - For columns, it performs a similar check.
- **Checking Diagonals:**
  - It checks both main diagonal (top-left to bottom-right) and the other diagonal (top-right to bottom-left).
- **Return Value:**
  - If a winner is found, it returns the winning symbol ('X' or 'O').
  - If no winner is found, it returns 'N'.

main Function:

- **Input:**
  - It declares a 3x3 character array box to represent the Tic Tac Toe board.
  - It uses scanf to read the input for each row of the board.
- **Calling checkWinner:**
  - It calls the checkWinner function with the input board to determine the winner.
- **Printing Result:**
  - It prints the result based on the return value from checkWinner.
  - If 'X' wins, it prints "X".
  - If 'O' wins, it prints "O".
  - If there is no winner, it prints "NO".

#CODE :

```
#include<stdio.h>
char checkWinner(char box[3][3])
{
   // Check rows
   for (int i = 0; i < 3; i++)
   {
      if (box[i][0] == box[i][1] && box[i][1] == box[i][2] && box[i][0] != '_')
      {
         return box[i][0];
      }
   }

    // Check columns
   for (int i = 0; i < 3; i++)
   {
      if (box[0][i] == box[1][i] && box[1][i] == box[2][i] && box[0][i] != '_')
      {
         return box[0][i];
      }
   }

   // Check diagonals
   if (box[0][0] == box[1][1] && box[1][1] == box[2][2] && box[0][0] != '_')
   {
      return box[0][0];
   }
   if (box[0][2] == box[1][1] && box[1][1] == box[2][0] && box[0][2] != '_')
   {
      return box[0][2];
   }


 // If no winner is found, return 'N' (for "NO")
   return 'N';
}

int main()
{
   char box[3][3];

   // Read the Tic Tac Toe field from input
   for (int i = 0; i < 3; i++)
   {
      scanf("%s", box[i]);
   }

   // Find the winner
   char winner = checkWinner(box);
```

```
    // Print the result
    if (winner == 'X')
    {
        printf("X\n");
    }
    else if (winner == 'O')
    {
        printf("O\n");
    }
    else
    {
        printf("NO\n");
    }
}
```

OUTPUT :

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Compile Message

Compile Time

```
Solution.c: In function ‘main’:
Solution.c:43:9: warning: implicit declaration of function ‘scanf’ [-Wimplicit-function-declaration]
         scanf("%s", box[i]);
         ^~~~
Solution.c:43:9: warning: incompatible implicit declaration of built-in function ‘scanf’
Solution.c:43:9: note: include ‘<stdio.h>’ or provide a declaration of ‘scanf’
Solution.c:1:1:
+#include <stdio.h>
 char checkWinner(char box[3][3])
Solution.c:43:9:
         scanf("%s", box[i]);
         ^~~~
Solution.c:52:9: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-declaration]
         printf("X\n");
         ^~~~~~
Solution.c:52:9: warning: incompatible implicit declaration of built-in function ‘printf’
Solution.c:52:9: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
Solution.c:56:9: warning: incompatible implicit declaration of built-in function ‘printf’
         printf("O\n");
         ^~~~~~
Solution.c:56:9: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
Solution.c:60:9: warning: incompatible implicit declaration of built-in function ‘printf’
         printf("NO\n");
         ^~~~~~
Solution.c:60:9: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
```

Input (stdin)

Run Time

```
OXO
XOX
OX_
```

Your Output (stdout)

```
0
```

Expected Output

```
0
```

Problem Statement: Dynamic Memory Allocation, Arrays, and Structures: Tic-Tac-Toe Game. In this programming assignment, you will implement a Tic-Tac-Toe game in C that utilizes dynamic memory allocation, arrays and structures .The game will be played between two players, 'X' and 'O' on a 3X3 grid. The program will dynamically allocate memory for the grid and use structures to represent the players and their moves.

INFERENCE :

struct Player:

- A structure is created to represent a player with a symbol (either 'X' or 'O') and a score.

initializeGrid Function:

- It dynamically allocates memory for a 3x3 game grid and initializes all cells with spaces.

printGrid Function:

- It prints the current state of the Tic-Tac-Toe grid, displaying the cells and separators to represent the board.

checkWin Function:

- It checks if the current player has won by examining rows, columns, and diagonals.

main Function:

- It initializes players (player X and player O) and the game grid using the functions described above.
- The game loop runs until there is a winner or a tie (9 moves).
- Inside the loop, it prints the current state of the grid, prompts the current player for their move, and validates the input.
- It updates the grid with the player's move and checks for a win. If a win is detected, it prints the winner and breaks out of the loop.
- If no winner is found after 9 moves, it declares a tie.
- Finally, it deallocates the memory used by the game grid.

Input Validation:

- The code includes input validation to ensure that the entered row and column values are within the valid range (0 to 2) and that the selected cell is not already occupied.
- It also handles invalid inputs, clearing the input buffer to avoid an infinite loop due to invalid input.

#CODE 2:

```
#include <stdio.h>
#include <stdlib.h>

// Structure to represent a player
struct Player {
    char symbol;
    int score;
};

// Function to dynamically allocate and initialize the game grid
char** initializeGrid() {
    char** grid = (char**)malloc(3 * sizeof(char*));
    for (int i = 0; i < 3; i++) {
        grid[i] = (char*)malloc(3 * sizeof(char));
        for (int j = 0; j < 3; j++) {
            grid[i][j] = ' ';
        }
    }
    return grid;
}

// Function to print the Tic-Tac-Toe grid
void printGrid(char** grid) {
    printf("\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf(" %c ", grid[i][j]);
            if (j < 2) {
                printf("|");
            }
        }
        printf("\n");
        if (i < 2) {
            printf("-----------\n");
        }
    }
    printf("\n");
}

// Function to check if the current player has won
int checkWin(char** grid, char symbol) {
    // Check rows and columns
    for (int i = 0; i < 3; i++) {
        if (grid[i][0] == symbol && grid[i][1] == symbol && grid[i][2] == symbol) {
            return 1; // Row win
        }
        if (grid[0][i] == symbol && grid[1][i] == symbol && grid[2][i] == symbol) {
            return 1; // Column win
```

```
        }
    }

    // Check diagonals
    if ((grid[0][0] == symbol && grid[1][1] == symbol && grid[2][2] == symbol) ||
        (grid[0][2] == symbol && grid[1][1] == symbol && grid[2][0] == symbol)) {
        return 1; // Diagonal win
    }

    return 0; // No win
}

int main() {
    // Initialize players
    struct Player playerX = { 'X', 0 };
    struct Player playerO = { 'O', 0 };

    // Initialize the game grid
    char** grid = initializeGrid();

    int currentPlayer = 0; // 0 for X, 1 for O
    int moves = 0;
    int row, col;

    while (moves < 9) {
        printGrid(grid);

        printf("Player %c, enter your move (row and column): ", currentPlayer == 0 ? 'X' : 'O');
        if (scanf("%d %d", &row, &col) != 2 || row < 0 || row > 2 || col < 0 || col > 2 ||
grid[row][col] != ' ') {
            printf("Invalid input. Please try again.\n");
            while (getchar() != '\n'); // Clear input buffer
            continue;
        }

        grid[row][col] = currentPlayer == 0 ? 'X' : 'O';
        moves++;

        if (checkWin(grid, playerX.symbol)) {
            playerX.score = 1;
            printGrid(grid);
            printf("Player X wins!\n");
            break;
        } else if (checkWin(grid, playerO.symbol)) {
            playerO.score = 1;
            printGrid(grid);
            printf("Player O wins!\n");
            break;
        }
```

```
            currentPlayer = 1 - currentPlayer; // Switch players
        }

        if (moves == 9) {
            printGrid(grid);
            printf("It's a tie!\n");
        }

        // Deallocate memory for the grid
        for (int i = 0; i < 3; i++) {
            free(grid[i]);
        }
        free(grid);

        return 0;
    }
```

OUTPUT :

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\mahes\Documents\c programing> cd "c:\Users\mahes\Documents\c programing\" ; if ($?) { gcc dsa.c
-o dsa } ; if ($?) { .\dsa }

   |   |
-----------
   |   |
-----------
   |   |

Player X, enter your move (row and column): 0 1

   | X |
-----------
   |   |
-----------
   |   |
Player O, enter your move (row and column): 0 0

 O | X |
-----------
   |   |
-----------
   |   |

Player X, enter your move (row and column): 1 1

 O | X |
-----------
   | X |
-----------
   |   |
Player O, enter your move (row and column): 2 2

 O | X |
-----------
   | X |
-----------
   |   | O

Player X, enter your move (row and column): 2 1

 O | X |
-----------
   | X |
-----------
   | X | O

Player X wins!
PS C:\Users\mahes\Documents\c programing>
```