

1. INTRODUCTION

Tic Tac Toe is a classic two-player strategy game that takes place on a 3×3 grid. In this game, players alternate turns marking the available cells with either an 'X' or an 'O'. The primary objective is to be the first to align three identical symbols in a row, column, or diagonal. Despite its simplicity, Tic Tac Toe effectively illustrates the principles of strategic planning, pattern recognition, and logical reasoning.

Owing to its well-defined rules and minimal complexity, Tic Tac Toe is frequently utilized as an introductory project for individuals beginning their programming journey. It provides a practical framework for understanding fundamental programming concepts such as data representation, control structures, and algorithmic logic.

This project presents the development of the Tic Tac Toe game using the C programming language. The implementation employs a two-dimensional array to represent the game board, ensuring efficient management of game states. Pointers are incorporated to demonstrate their role in accessing and modifying elements within the array, thereby enhancing the program's efficiency. Furthermore, the project highlights the application of essential programming constructs, including loops, conditional statements, and user input handling in C.

Ultimately, this project not only recreates a familiar and interactive game but also serves as an educational tool for reinforcing core programming principles and improving problem-solving skills within a structured coding environment.

2. ALGORITHM

1.Start

2.Initialize board

Declare a 3×3 character array board and initialize all positions with blank space ' '.

Set player = 1, moves = 0, win = 0.

3.Repeat forever (main game loop)

3.1 Display a blank line.

3.2 Print the current board in 3 rows and 3 columns:

For each row i from 0 to 2

For each column j from 0 to 2

Print the value at board[i][j] using pointer notation $*(*(board + i) + j)$.

If $j < 2$, print column separator "|".

If $i < 2$, print row separator "----+----+----".

4.Set current player's mark

If player == 1, set mark = 'X'.

Else set mark = 'O'.

5.Input move

Prompt: "Player player (mark), enter row and column (1–3 each):".

Read row and col.

Decrement both: row-- and col-- to convert to 0-based indices.

6. Validate input

If row or col is outside 0–2:

Print “Invalid input, try again.”

Use continue to skip to the next iteration of the loop.

Else if the cell $*(\text{board} + \text{row}) + \text{col}$ is not a space ' ':

Print “Cell taken, try again.”

Use continue to skip to the next iteration.

7. Place mark

Assign mark to the chosen cell: $*(\text{board} + \text{row}) + \text{col} = \text{mark}$.

Increment moves

8. Check for win in rows and columns

For each index i from 0 to 2:

Row check:

If $\text{board}[i][0]$, $\text{board}[i][1]$, and $\text{board}[i][2]$ (via pointer notation) are all equal to mark, set $\text{win} = 1$ and break the loop.

Column check:

If $\text{board}[0][i]$, $\text{board}[1][i]$, and $\text{board}[2][i]$ are all equal to mark, set $\text{win} = 1$ and break the loop.

9. Check diagonals for win

- If $\text{board}[0][0]$, $\text{board}[1][1]$, and $\text{board}[2][2]$ are all equal to mark, set $\text{win} = 1$.
- Else if $\text{board}[0][2]$, $\text{board}[1][1]$, and $\text{board}[2][0]$ are all equal to mark, set $\text{win} = 1$.

10. If a win is detected

If $\text{win} == 1$:

Print “Player player (mark) wins!”.

Print the final board in the same format as step 3.2.

break from the main loop.

11.Check for tie

If moves == 9 and win == 0:

Print “It’s a tie! Both players are equal on their inputs.”

Print the final board.

break from the main loop.

12.Switch player

If player == 1, set player = 2.

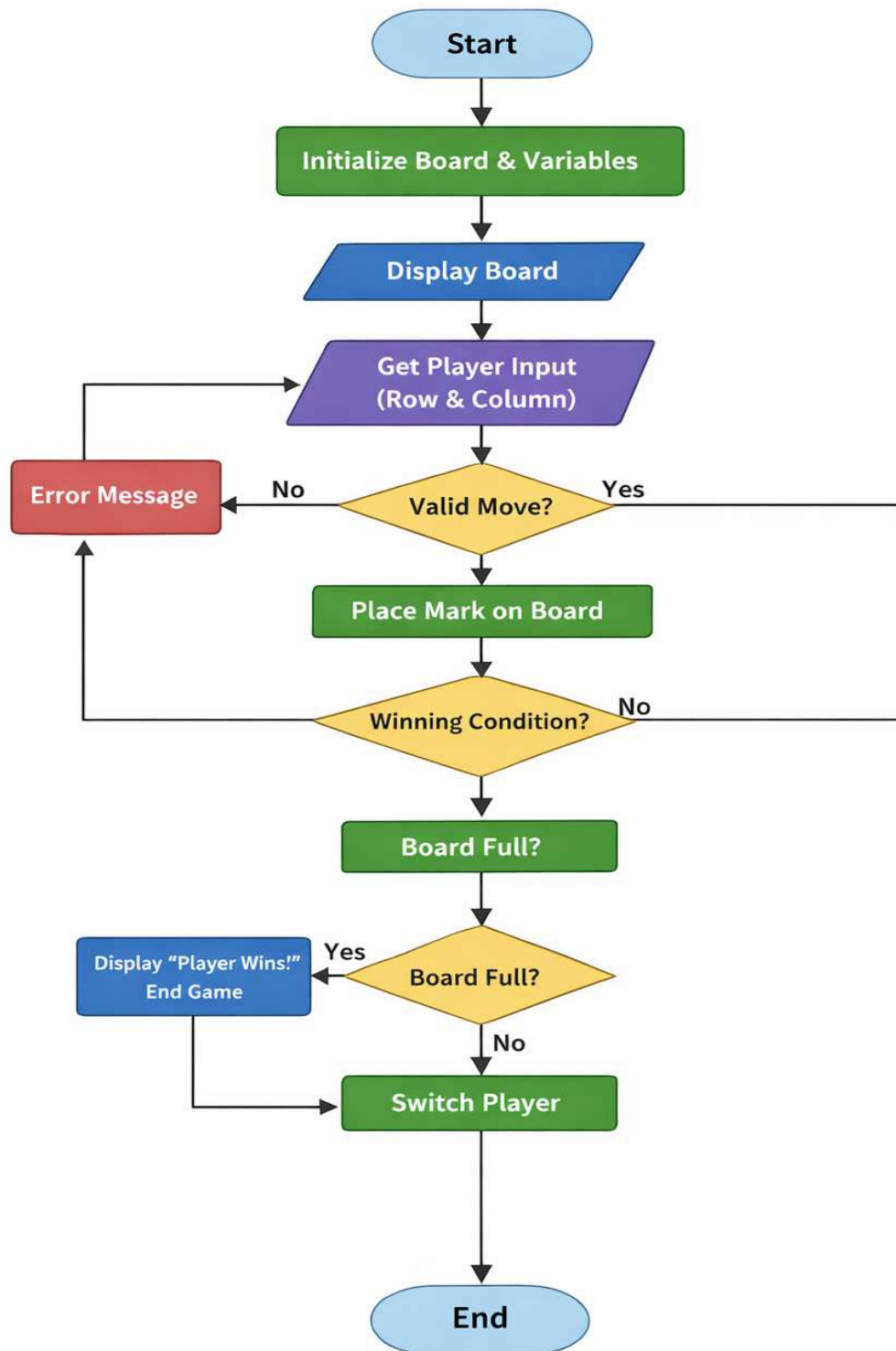
Else set player = 1.

Continue the loop to the next turn.

13.End

After breaking out of the loop, return 0 and stop the program.

3. FLOWCHART



4. SOURCE CODE

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

```
int main() {
```

```
    char board[3][3] = {
```

```
        {' ', ' ', ' '},
```

```
        {' ', ' ', ' '},
```

```
        {' ', ' ', ' '}
```

```
    };
```

```
    int player = 1;
```

```
    int moves = 0;
```

```
    char mark;
```

```
    int win = 0;
```

```
    int row, col;
```

```
    // Main game loop
```

```
    while (1) {
```

```
        // Print current board
```

```
        printf("\n");
```

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

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

```
                printf(" %c ", (*(board + i) + j)); // Pointer notation
```

```
                if (j < 2) printf("|");
```

```
            }
```

```
        if (i < 2) printf("\n---+---+---\n");
    }

    printf("\n");

    // Set player's mark

    if (player == 1) mark = 'X';

    else mark = 'O';

    // Player input: Get row and column

    printf("Player %d (%c), enter row and column (1-3 each): ", player, mark);

    scanf("%d %d", &row, &col);

    // Adjust indices: C arrays start from 0

    row--; col--;

    // Input validation

    if ((row < 0 || row > 2) || (col < 0 || col > 2)) {

        printf("Invalid input, try again.\n");

        continue; // Jump statement

    }

    if (*(*(board + row) + col) != ' ') {

        printf("Cell taken, try again.\n");

        continue; // Jump statement
```

```
}
```

```
// Place mark and count move
```

```
*(*(board + row) + col) = mark;
```

```
moves++;
```

```
// Check for win (rows, columns, diagonals)
```

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

```
    // Row
```

```
    if (*(*(board + i) + 0) == mark &&
```

```
        (*(board + i) + 1) == mark &&
```

```
        (*(board + i) + 2) == mark) {
```

```
        win = 1;
```

```
        break; // Jump statement
```

```
    }
```

```
    // Column
```

```
    if (*(*(board + 0) + i) == mark &&
```

```
        (*(board + 1) + i) == mark &&
```

```
        (*(board + 2) + i) == mark) {
```

```
        win = 1;
```

```
        break; // Jump statement
```

```
    }
```

```
}
```

```
// Diagonals
```

```
if (*(*(board + 0) + 0) == mark &&
```

```
    (*(*(board + 1) + 1) == mark &&
```

```
    (*(*(board + 2) + 2) == mark) {
```

```
    win = 1;
```

```
}
```

```
else if (*(*(board + 0) + 2) == mark &&
```

```
    (*(*(board + 1) + 1) == mark &&
```

```
    (*(*(board + 2) + 0) == mark) {
```

```
    win = 1;
```

```
}
```

```
// If win is detected:
```

```
if (win == 1) {
```

```
    printf("\nPlayer %d (%c) wins!\n", player, mark);
```

```
    // Show final board
```

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

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

```
            printf(" %c ", (*(board + i) + j));
```

```
            if (j < 2) printf("|");
```

```
        }
```

```
        if (i < 2) printf("\n---+---+---\n");
```

```
    }
```

```
    printf("\n");

    break; // Ends game
}

// Check for tie (board full, no win)
if (moves == 9 && win == 0) {

    printf("\nIt's a tie! Both players are equal on their inputs.\n");

    // Print final board

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

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

            printf(" %c ", (*(board + i) + j));

            if (j < 2) printf("|");

        }

        if (i < 2) printf("\n---+---+---\n");

    }

    printf("\n");

    break; // Ends game
}

// Switch player
if (player == 1) player = 2;

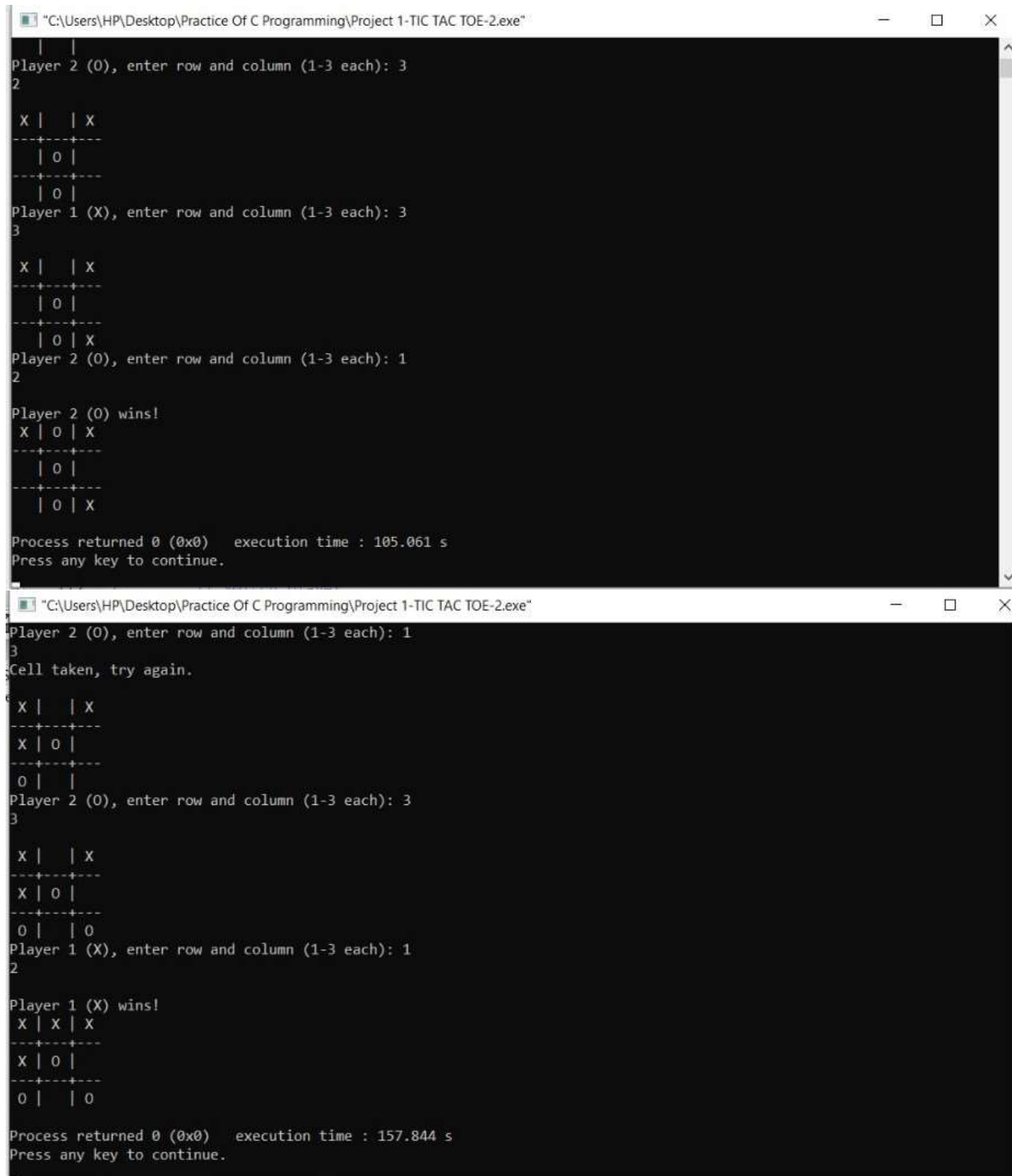
else player = 1;

}
```

```
return 0;
```

```
}
```

5. RESULTS



```
"C:\Users\HP\Desktop\Practice Of C Programming\Project 1-TIC TAC TOE-2.exe"
|
|
Player 2 (O), enter row and column (1-3 each): 3
2
X |  | X
---+---
| O |
---+---
| O |
---+---
Player 1 (X), enter row and column (1-3 each): 3
3
X |  | X
---+---
| O |
---+---
| O | X
---+---
Player 2 (O), enter row and column (1-3 each): 1
2
Player 2 (O) wins!
X | O | X
---+---
| O |
---+---
| O | X
---+---
Process returned 0 (0x0)   execution time : 105.061 s
Press any key to continue.

"C:\Users\HP\Desktop\Practice Of C Programming\Project 1-TIC TAC TOE-2.exe"
Player 2 (O), enter row and column (1-3 each): 1
3
Cell taken, try again.
X |  | X
---+---
X | O |
---+---
O |  |
---+---
Player 2 (O), enter row and column (1-3 each): 3
3
X |  | X
---+---
X | O |
---+---
O |  | O
---+---
Player 1 (X), enter row and column (1-3 each): 1
2
Player 1 (X) wins!
X | X | X
---+---
X | O |
---+---
O |  | O
---+---
Process returned 0 (0x0)   execution time : 157.844 s
Press any key to continue.
```

"C:\Users\HP\Desktop\Practice Of C Programming\Project 1-TIC TAC TOE-2.exe"

```
X | O |  
Player 1 (X), enter row and column (1-3 each): 3
```

3

```
X | | O  
+---+  
O | | X  
+---+  
X | O | X
```

```
O | | X
```

```
X | O | X
```

```
Player 2 (O), enter row and column (1-3 each): 2
```

2

```
X | | O  
+---+  
O | O | X  
+---+  
X | O | X
```

```
O | O | X
```

```
X | O | X
```

```
Player 1 (X), enter row and column (1-3 each): 1
```

2

It's a tie! Both players are equal on their inputs.

```
X | X | O  
+---+  
O | O | X  
+---+  
X | O | X
```

```
O | O | X
```

```
X | O | X
```

```
X | O | X
```

Process returned 0 (0x0) execution time : 53.590 s

Press any key to continue.

6. OUTCOMES

By completing this project, the following outcomes are achieved:

I. Strong Understanding of Two-Dimensional Arrays in C

You will gain a clear understanding of how two-dimensional arrays work, including their declaration, initialization, and manipulation. This will help you organize and manage data efficiently within a program.

II. Practical Skills with Pointers and Pointer Arithmetic

The project offers hands-on practice with pointers, allowing you to understand how memory addresses work and how to perform pointer arithmetic. You will also see how pointers and arrays are closely related, improving your understanding of how data is stored and accessed in C.

III. Improved Use of Loops and Conditional Statements

Through the project's logic and structure, you will strengthen your ability to use loops and conditionals effectively. This will help you write clear, logical, and efficient code that can handle different scenarios within your program.

IV. Experience with Input Validation

You will learn how to apply input validation techniques to ensure the program handles user input correctly. This experience will help you build more reliable and user-friendly applications.

V. Hands-On Development of an Interactive Game in C

The project concludes with creating a simple, interactive game. This practical task allows you to combine all the concepts you've learned—arrays, pointers, loops, and input validation—into a complete working program.

7. CONCLUSION

The Tic Tac Toe game has been successfully developed using the C programming language, demonstrating a practical application of several key programming concepts. Through this project, two-dimensional arrays were effectively utilized to represent and manage the game board, allowing for efficient data storage and manipulation. The use of pointers further enhanced the program's flexibility by enabling direct access and modification of memory, showcasing one of C's most powerful features.

Additionally, the implementation of loops and conditional statements played a vital role in controlling the flow of the game, handling player turns, and determining winning conditions. Input validation techniques ensured that the game could handle incorrect or unexpected inputs gracefully, improving both its reliability and user experience.

Overall, the project not only resulted in a functional and interactive game but also strengthened understanding of core programming principles such as data handling, logical structuring, and control flow. This hands-on experience serves as a valuable step toward building more complex applications in C and enhances problem-solving and analytical thinking skills.

REFERENCES

- <https://github.com/abhixsh/Tic-Tac-Toe-Game>
- <https://itsourcecode.com/free-projects/c-projects/tic-tac-toe-in-c-programming-with-source-code/>