

# Lab 5: Creating a Tic-Tac-Toe Game Objective:

To create a simple Tic-Tac-Toe game in C where two players can play against each other. The game will track wins, losses, and draws, and allow players to play multiple rounds.

# Step 1: Setting Up the Environment

1. **Create a new C project** in your preferred IDE or text editor.
2. **Create a new file** named main.c.

# Step 2: Writing the Main Function

1. **Include necessary headers**:



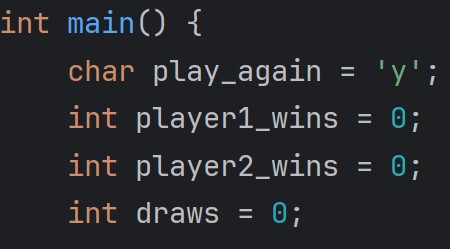
* **#include <stdio.h>:** This header file is included to use standard input and output functions like printf and scanf.
* **#include <stdlib.h>:** This header file is included to use functions like malloc and free for dynamic memory allocation.
* **#include "tic\_tac\_toe.h":** This is a custom header file that contains the declarations for the Tic-Tac-Toe game functions and structures.

# Define the main function:

* **char play\_again = 'y';:** This variable keeps track of whether the user wants to play another game.



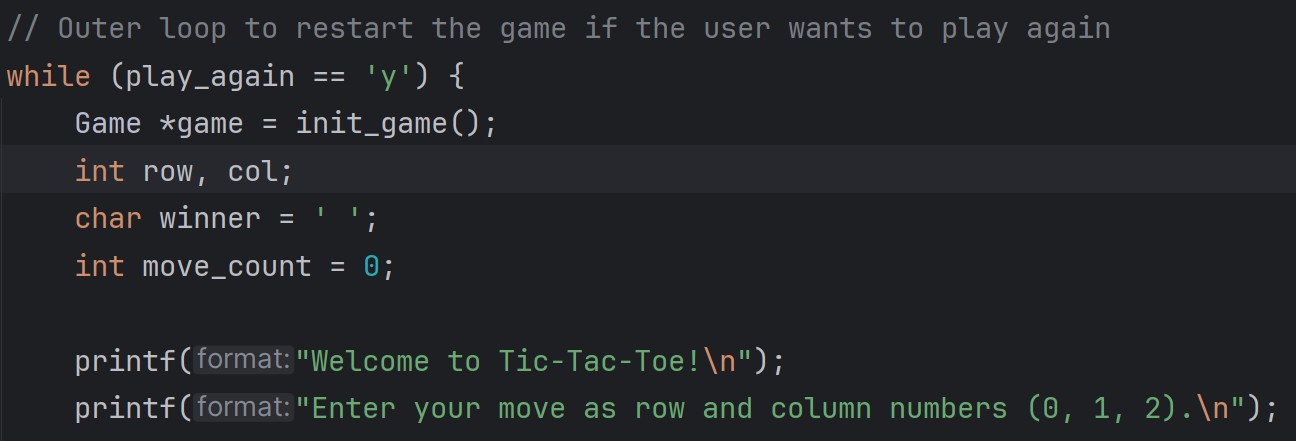
* **int player1\_wins = 0;:** This variable counts the number of wins for Player 1.
* **int player2\_wins = 0;:** This variable counts the number of wins for Player 2.
* **int draws = 0;:** This variable counts the number of draws.



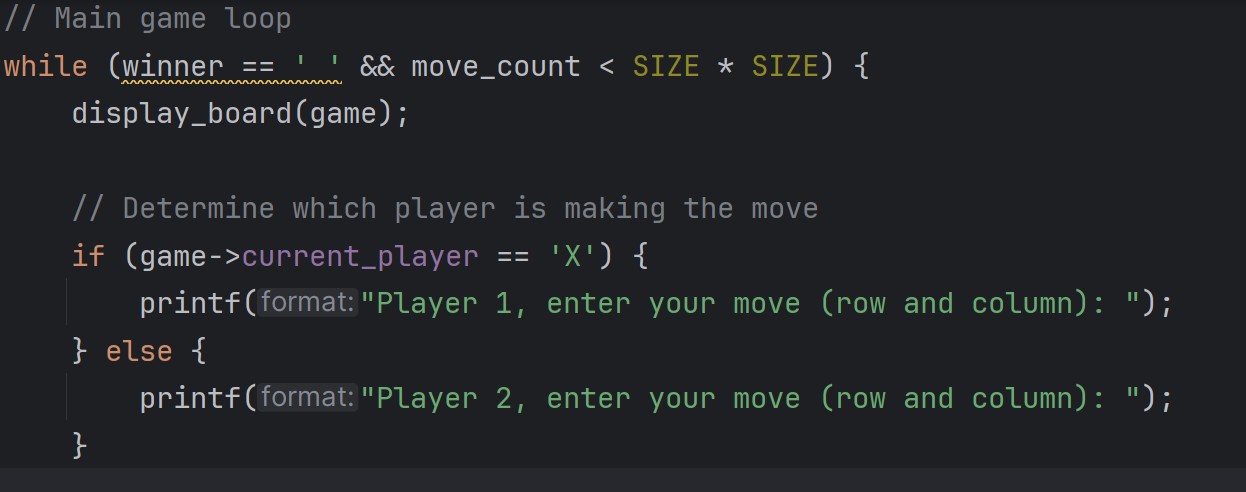
The loop while (play\_again == 'y') allows the game to restart if the user wants to play again. Inside this loop, Game \*game = init\_game(); initializes a new game by calling the init\_game function, which sets up the game board and the current player. The variables int row, col; store the row and column of the player’s move, while char winner = ' '; is used to store the winner of the game, starting as a space character to indicate no winner yet.

The variable int move\_count = 0; counts the number of moves made in the game. The statement printf("Welcome to Tic-Tac-Toe!\n"); prints a welcome message, and printf("Enter your move as row and column numbers (0, 1, 2).\n"); instructs the players on how to enter their moves.

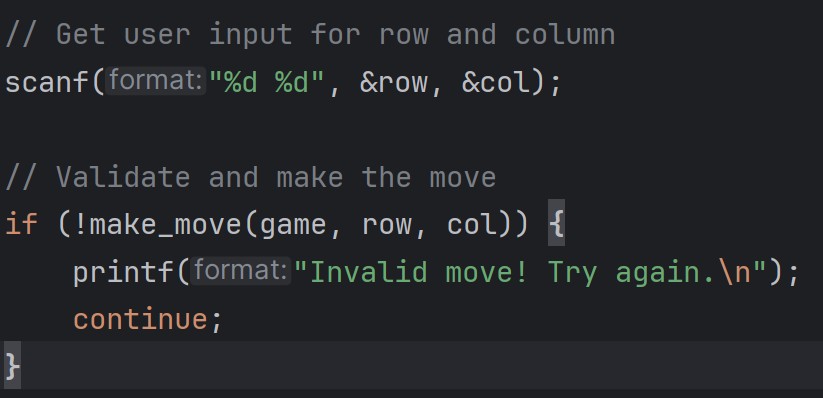


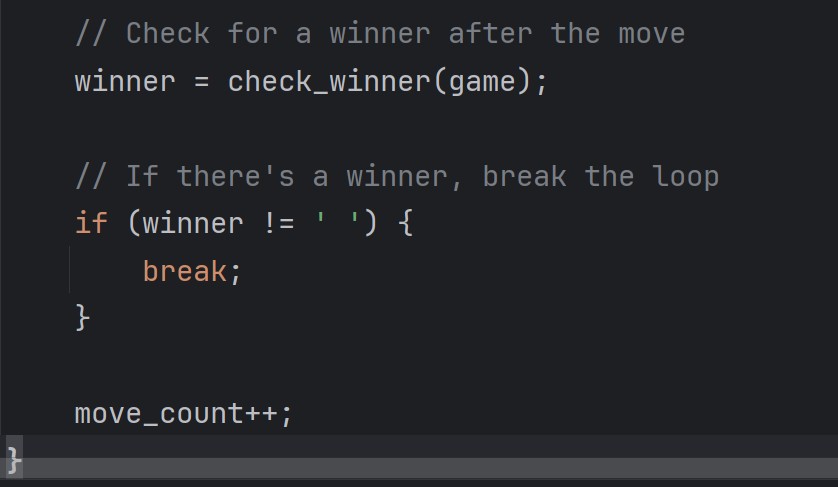


The loop while (winner == ' ' && move\_count < SIZE \* SIZE) continues until there is a winner or the board is full. Inside this loop, display\_board(game); is called to display the current state of the game board. The statement if (game->current\_player == 'X') checks which player’s turn it is and prompts them to enter their move. The function scanf("%d %d", &row, &col); reads the row and column input from the player. The function if (!make\_move(game, row, col)) attempts to make the move; if the move is invalid, it prints an error message and continues the loop. The function winner = check\_winner(game); checks if there is a winner after the move. Finally, move\_count++; increments the move count.









The function display\_board(game); displays the final state of the game board. The conditional statement if (winner == 'X') { checks if Player 1 has won and increments their win count.

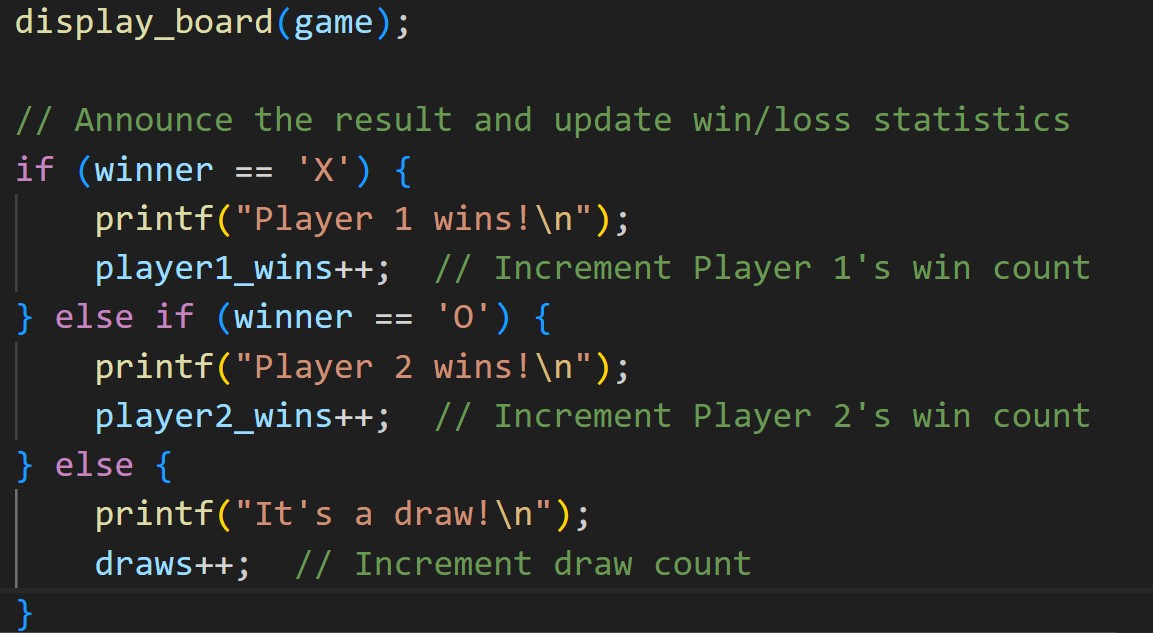


Similarly, else if (winner == 'O') { checks if Player 2 has won and increments their win count. If neither player has won, else { checks if the game is a draw and increments the draw count.

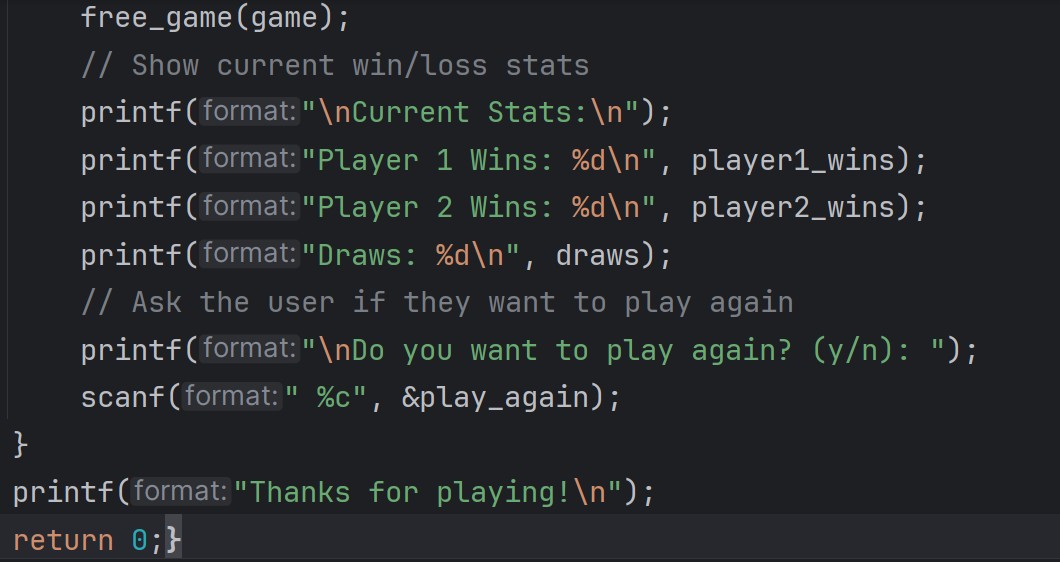
The function free\_game(game); frees the memory allocated for the game. The statement printf("\nCurrent Stats:\n"); prints the current win/loss statistics, followed by printf("Player 1 Wins: %d\n", player1\_wins); which prints the number of wins for Player 1, printf("Player 2 Wins: %d\n", player2\_wins); which prints the number of wins for Player 2, and printf("Draws: %d\n", draws); which prints the number of draws.

The prompt printf("\nDo you want to play again? (y/n): "); asks the user if they want to play again, and scanf(" %c", &play\_again); reads the user’s input, ensuring any previous newline character is skipped.

Finally, printf("Thanks for playing!\n"); prints a thank you message when the game ends, and return 0; returns 0 to indicate that the program ended successfully.







# Step 3: Implementing the Game Logic

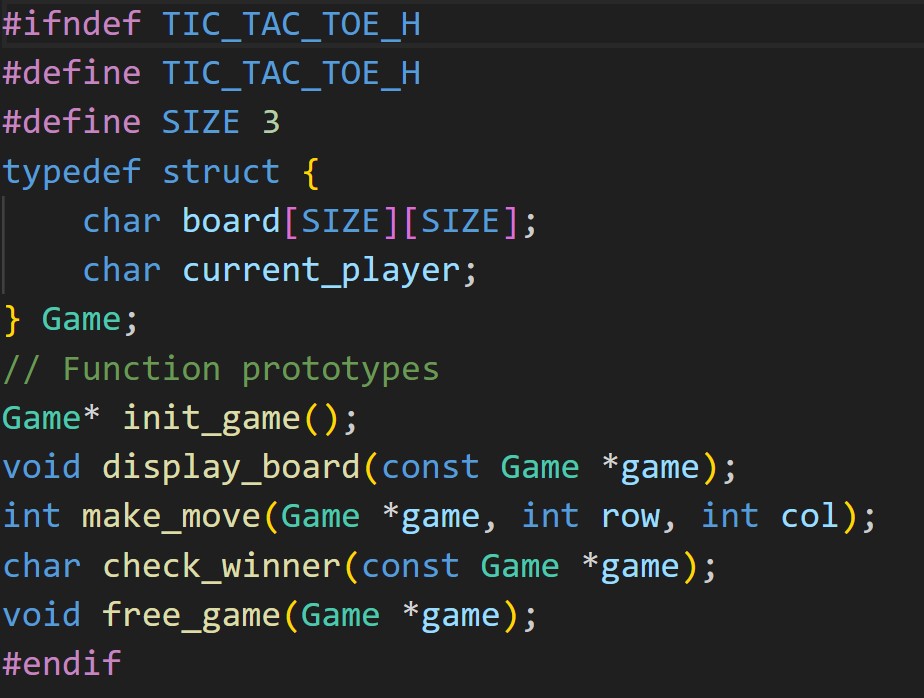
1. **Create a new file** named tic\_tac\_toe.h and tic\_tac\_toe.c.
2. **Define the game structure and functions** in tic\_tac\_toe.h

The header file begins with #ifndef TIC\_TAC\_TOE\_H and #define TIC\_TAC\_TOE\_H to prevent multiple inclusions of the same header file, which can cause compilation errors. The macro #define SIZE 3 sets the size of the Tic-Tac-Toe board to 3x3. The typedef struct { ... } Game; defines a structure named Game that contains a 2D array board to represent the game board and a current\_player variable to track whose turn it is.

The function prototypes declared include Game\* init\_game(); for initializing a new game, void display\_board(const Game \*game); for displaying the current state of the game board, int make\_move(Game \*game, int row, int col); for making a move on the board, char check\_winner(const Game \*game); for checking if there is a winner, and void free\_game(Game

\*game); for freeing the memory allocated for the game. The header file ends with #endif to close the conditional inclusion.





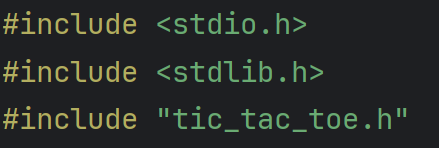
1. **Implement the functions** in tic\_tac\_toe.c:

#include <stdio.h>: This header file is included to use standard input and output functions like printf and scanf.

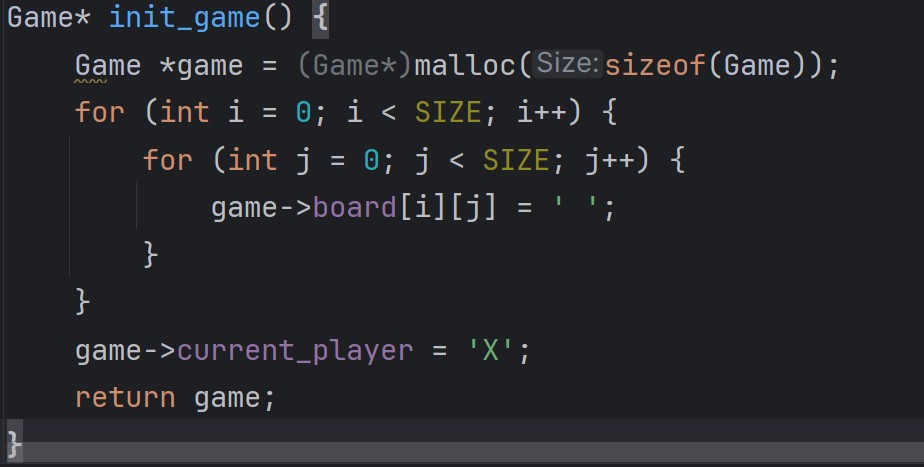
#include <stdlib.h>: This header file is included to use functions like malloc and free for dynamic memory allocation.

#include "tic\_tac\_toe.h": This is a custom header file that contains the declarations for the Tic- Tac-Toe game functions and structures.





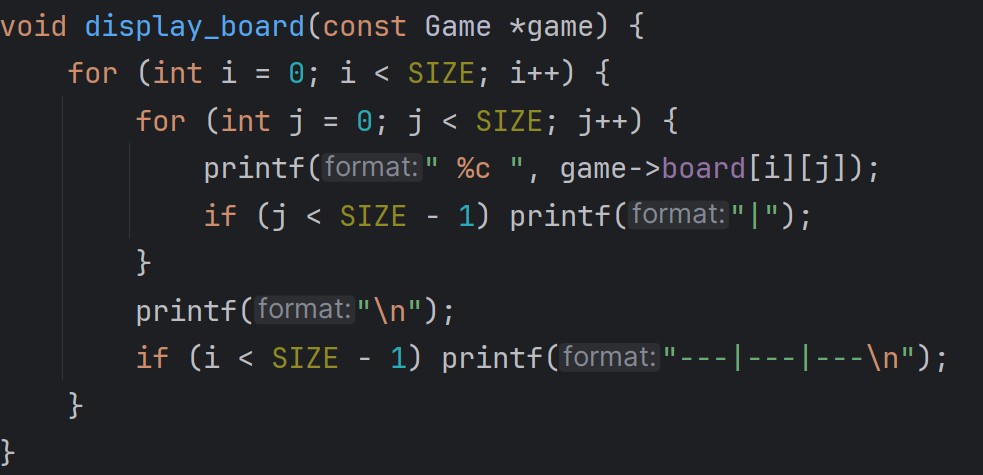
The function Game\* init\_game() initializes a new game by allocating memory for a Game structure using malloc. It then sets up the game board to be empty by iterating through each cell and setting it to a space character. The starting player is set to ‘X’ with game->current\_player = 'X';. Finally, the function returns a pointer to the newly initialized game structure.



The function void display\_board(const Game \*game) displays the current state of the game board. It iterates through the board using nested loops, printing each cell’s content. For each cell, it prints the character stored in the board array. If the cell is not the last in its row, it prints a vertical line

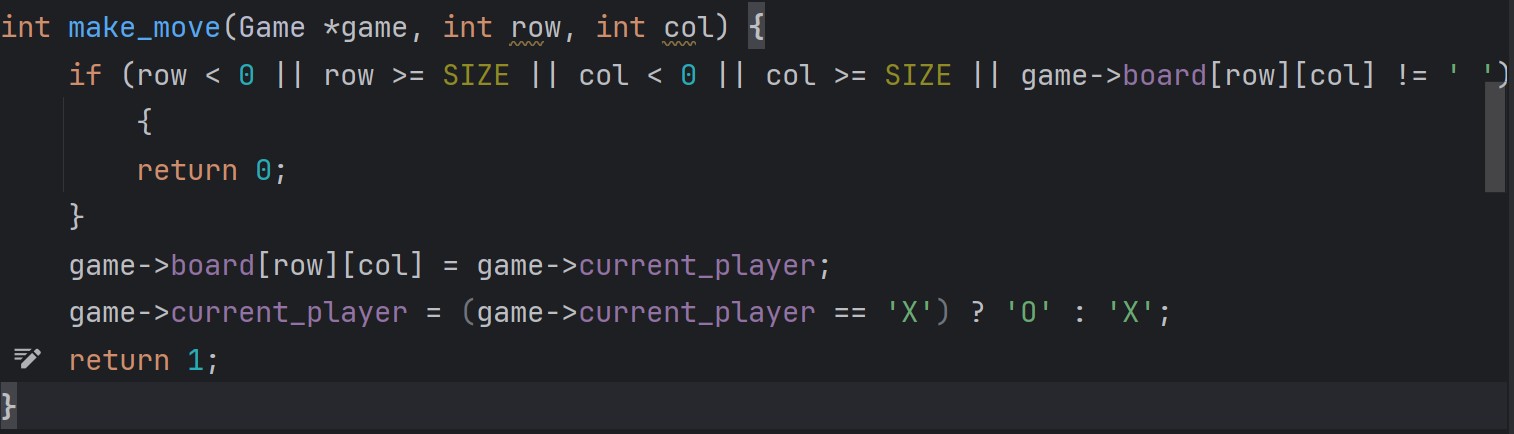


to separate the cells. After each row, it prints a newline character. If the row is not the last, it prints a horizontal line to separate the rows, creating a visual representation of the Tic-Tac-Toe board.

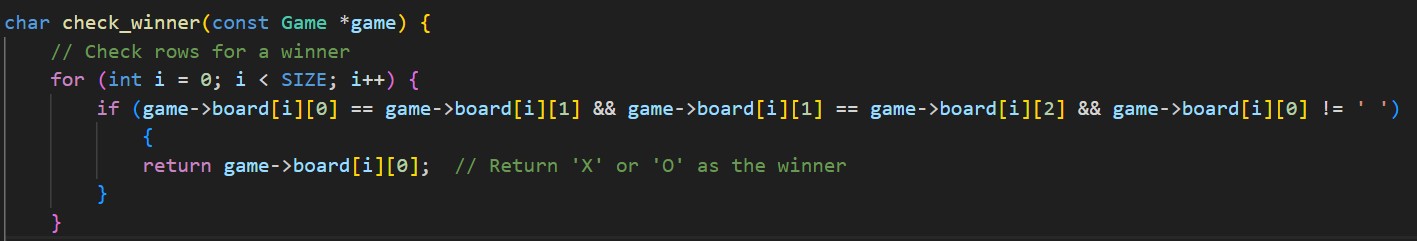


The function int make\_move(Game \*game, int row, int col) makes a move on the board. It first checks if the move is valid by ensuring the row and column are within bounds and the cell is empty. If any of these conditions are not met, it returns 0, indicating an invalid move. If the move is valid, it places the current player’s mark on the board at the specified row and column. It then switches the current player from ‘X’ to ‘O’ or vice versa. Finally, it returns 1 to indicate a valid move.





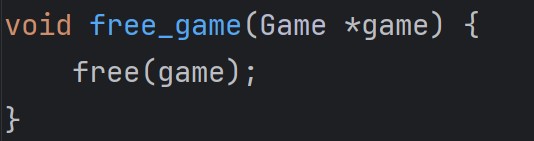
The function char check\_winner(const Game \*game) checks for a winner by iterating through each row, column, and diagonal. It returns the winner (‘X’ or ‘O’) if all three cells in a row, column, or diagonal are the same and not empty. If no winner is found, it returns a space character ’ ’ to indicate no winner.







The function void free\_game(Game \*game) is responsible for freeing the memory allocated for the game. It does this by calling the free(game); function, which deallocates the memory that was previously allocated for the Game structure, ensuring there are no memory leaks.



# Step 4: Compiling and Running the Program

Create a Makefile using the structure provided in class. This Makefile should compile and link your Tic-Tac-Toe game, and include rules for cleaning up the build files.

# Part 2: Tasks to Solve and Submit Step 5: Enhancing the Game

1. **Add a feature to track the number of moves made by each player**:
   * Modify the Game structure to include counters for each player’s moves.
   * Update the make\_move function to increment these counters.
   * Display the move counts at the end of each game.

# Allow multiple players to play a game (bonus task):

* + Allow any number of players to participate in the game, not just two. Each player will be assigned a unique symbol (e.g., 'X', 'O', 'A', 'B', etc.).
  + Instead of switching between two players, use a system to track the current player's turn and rotate through the list of players.
  + Ask the user how many players will participate and adjust the turn-based system accordingly.

# Submission Requirements



Students should submit the following files:

1. **main.c**: Contains the main game loop, handles user input, and manages the game state.
2. **tic\_tac\_toe.h**: Header file that declares the game structure and functions.
3. **tic\_tac\_toe.c**: Implements the game logic, including initializing the game, displaying the board, making moves, checking for a winner, and freeing memory.
4. **Makefile**: Automates the compilation and linking process and includes rules for cleaning up the build files.
5. **Screenshots**: Include screenshots of the game running, showing different states such as the initial board, a few moves, and the end of the game with the result.

Make sure all files are properly commented and organized.

# Marking Scheme:

1. **Part 1: Completing the Game Setup (8 Marks)**:
   * This part includes:
     + Setting up the Tic-Tac-Toe game.
     + Allowing the game to support two players.
     + Allow user to input their move in 3x3 board.
     + Handling player turns and determining a winner or draw.
   * **Marks Allocated**: 8 marks for successfully completing this part.

# Part 2: Tracking the Number of Moves (2 Marks):

* + This part adds a new feature to the game:
    - Tracking the number of moves made by each player.
    - Displaying the move counts at the end of each game.
  + **Marks Allocated**: 2 marks for implementing this feature correctly.

# Bonus: Additional 2 Marks:

* Allowing the game to support multiple players.
* Allowing users to input the number of players (between 2 to 8).



* + **Marks Allocated**: 2 bonus marks (not counted immediately but can be adjusted later).