---

**System Details**
MacOS, Xcode compiler C++20

---

**Program Summary**
Program outputs an interactive two-player TicTacToe game. This program emphasizes OOP, pointers, and constant definitions

---

**How I will explain the program**

- Below, I will explain the logic and technicalities of the program files.
- I explain the logic and code of the program going in a line to line fashion
- For classes, I explain their data member, and then their member functions

---

**main.cpp**
Driver construct two player objects with their usernames. This program allows the creation of multiple players that can play the game between each other.

We then construct a game object, which receives two players.The game object is initialized in the heap. At this point we use the access operator to call the game's function "run_game()".

---

**Player.hpp and Player.cpp**
The player class has 3 private data members, its username, the number of games won, and a variable that will hold the game symbol assigned for the current game (either X or O).

The class member functions are as follow:

- We use a list constructor and initialize the object with the given username.
- The rest of the member functions are self-explanatory. Although it's worth mentioning that the program takes full advantage of constant assignments, by ensuring that if a function doesn't need to, it returns a constant value,  defines the function as constant, or both.

---

**Game.hpp and Game.cpp**
The game class has 3 private data members. A 3x3 array holding the game's grid, a counter holding the number of rounds for the current game ( round is defined as when both players make a single mark on the board, in other words, one X and one O is marked), and an array of 2 pointers which point to the 2 current players of the game.

The class member functions are as follow:

- In the list constructor, we initiate 2 pointers for the players, and store them in the pointer array. This will be helpful when we are manipulating the player in multiple functions later on, as we can utilize the original player, instead of creating copies.
- print_grid() is defined as a constant function. It will print the current 3x3 grid as the game goes on.
- tick_grid() receives two constant parameter addresses, the cell and the current player. This function will tick/mark the grid based on the cell handed and the current player's symbol (being X or O).
- check_win() will return a constant integer, and it is defined as a constant function. This function receives the current player, and checks the grid for a horizontal, vertical, or diagonal win. We return 1 to indicate a win has been found.
- restart_game() returns a constant character, and it is defined as a constant function. This function outputs both player's current number of wins, and asks the user if they want to play another game.
- run_game(). This function will execute the bulk of the overall operation/game. We first enter a while loop based on the "play_game" condition. The program will continue playing game cycles unless the user changes this condition.

  Inside of the loop, we re-initialize the number of rounds and the grid to their defaults. Each new game we output the usernames of both players and print the default grid. At this point we assign the player a game symbol, either X or O.

  Now, we enter the loop that will run a singular game. We set the condition at i<6 since tictactoe's grid has 9 squares, and player alternate marks, a user has a maximum of 5 marks, thus, we exit the loop and end the current game if the number of turns reaches 5.

  Each round, we alternate between player1 and player2, where we ask the player which cell they want to mark, and call tick_cell(), print the updated grid, print_grid(), and check for a win with check_win().

  If a winner is found, we assign a win to that player, output the total number of wins both players have, and ask the user if they want to play another game with restart_game(). Note that if the player wants to exit the game, we execute i+=6, and j+=2. Recall that a single game has a max of 6 marks per player, so we add 6 to i to negate the condition of the loop and exit the current game. We also use j to alternate between player1 and 2, so we add 2 to j to negate the condition, since if player 1 won, we don't loop to loop to player 2 thereafter.

  However, if there is now winner yet, we add 1 to round, to keep track of the number of rounds. If the number of round reaches 6, we also end current game, as previously mentioned, and ask user if they want to play another game with restart_game();