STUDENT ID:
TOTAL MARKS:
WHOLE DAYS LATE (will be calculated)

PART 1: GENERALISING THE GAME (60%):

Despite having all the right parts you cannot get full marks unless
the marker can run. Test your code on the ECS computers.

No graceful error handling required.

__ main
- passes correctly by value (where appropriate)
- passes correctly by reference (where appropriate)

__ init_game(TicTacToe* game, int size)
- sets the size of the board
- uses malloc to allocate sufficient bytes to hold the board
- uses malloc to allocate so that can be treat as a 2D array
  (remember that malloc allocates non-contiguious memory)
- initialises the board to NONE

__ free_game(TicTacToe* game)
- deallocates memory correctly
- removes pointer to board

__ player_move(TicTacToe* game)
- correctly references the board member
- minimal changes to code, i.e. still references board as 2D array

__ computer move_(TicTacToe* game)
- correctly references the board member
- minimal changes to code, i.e. still references board as 2D array

__ print_game(TicTacToe game)
- correctly references the board member
- minimal changes to code, i.e. still references board as 2D array

__ check(TicTacToe* game)
- correctly references the board member
- treats the board member as a 2D array
- checks for N tokens in a row rather than fixed number
  (rows, columns, diagonals)
- correctly updates the winner member

__ print_result(TicTacToe game)
- checks the winner member to determine winner
- minimal changes to the code

PART 2: EXPERIMENT WITH UNIX PIPES (5%)

Despite having all the right parts you cannot get full marks unless
the marker can run. Test your code on the ECS computers.

No graceful error handling required.

__ Writer sends "Hi" to reader
- opens fifo pipe created by the reader
- sends message ("Hi") to reader via pipe
- creates a fifo pipe for reader to send data to writer
- blocks waiting to read message from reader
- prints out received message ("Hello")
- closes both fifo pipes
- removes the fifo pipe created by the writer when done

__ Reader sends "Hello" to writer
- creates a fifo pipe for writer to send data to readers
- opens fifo pipe created by the writer
- blocks waiting to read message from writer
- prints out received message ("Hi")
- sends message ("Hello") to the writer via the pipe
- closes both fifo pipes
- removes the fifo pipe created by the reader

PART 3: USE UNIX PIPES TO CREATE A CLIENT-SERVER SOLUTION (25%)

Despite having all the right parts you cannot get full marks unless
the marker can run. Test your code on the ECS computers.

No graceful error handling required.

CLIENT:

__ main method
- send the server the size of the game board.
- loop until the server tells us the game is over
  - get the player's move from console
  - send to server
  - repeats getting move and sending until server says alid move
  - wait until the server tells us result
- exits loop if server tells us game is over
- tidies up correctly (closes open pipes and delete file system)

__ init_game(int serverfd)
- ask the player
- send player's move to server
- returns size of board

__ player_move(int clientfd, int serverfd)
- ask the player for the move
- send the server the move (x,y ints)
- check the result

__ print_game(int clientfd, int size)
- prints out the board correctly

__ check(int clientfd)
- read the result that the server has placed in the client's pipe
- return result from function

__ print_winner(int clientfd)
- read the result
- prints out result to the console

SERVER:

__ main method
- read size of board from client
- intialise the board
- loop until either client or computer wins
  - loop reading the client's move until valid move received
  - make computer's move
- send result to client
- tidies up correctly (closes open pipes and delete file system)

__ init_game(TicTacToe* game,int size)
- correctly initialises the structure (as for part 1)

__ free_game(TicTacToe* game)

- corrrectly deallocates memory (as for part 1)

__ get_player_move(int serverfd, int clientfd, TicTacToe* game)
- reads move from client one int as a time via server's pipe
- return result to client via the client's pipe

__ computer_move(TicTacToe* game)
- correctly implements computer move (as for part 1)

__ print_game(int clientfd, TicTacToe game)
- correctly implements print game (as for part 1)

__ check(TicTacToe* game)
- correctly implements check (as for part 1)

__ void print_result(int clientfd, TicTacToe game)
- sends the result to the client via the client's pipe
  (sends an int)

__ void client_continue(int clientfd)
- tells the client that it should continue playing
  (sends an int)