**C PROGRAMMING - ASSIGNMENT 2 - X's and Os**

The goal of assignment 2 is to write a version of the game X's and Os.

The program will be started with two command line arguments specifying the names of the players.

play\_game John Mary

A blank 3 \* 3 game board is displayed at the start. Initially all the slots are empty.

**GAME STATUS**

**---------------------**

**- | - | -**

**----- ----- -----**

**- | - | -**

**----- ----- -----**

**- | - | -**

The first player is then prompted to play. The first player is randomly assigned and places X on the board. A board shows the numbers 0 to 8 that are used to identify each slot and the player must specify in which slot he wishes to place his mark. If the user chooses a slot that has been used he is prompted to enter an alternative slot.

**0 | 1 | 2**

**----- ----- -----**

**3 | 4 | 5**

**----- ----- -----**

**6 | 7 | 8**

**Mary's Turn : Enter a slot:** 0

Each time a player takes a turn the game board is displayed showing its current state.

**X | - | -**

**----- ----- -----**

**- | - | -**

**----- ----- -----**

**- | - | -**

The program must check after each player's turn to see

* If a line of X's or O's exist. If so the game ends and the winner is named.
* If all the slots have been used. If so the game ends and a draw is declared.

If the game is not over the play alternates between both players.

**Implementation**

To represent the “logic” of our game we use the following structure:

struct game {

char board[3][3];

char playerNames[2][MAX\_NAME\_LEN];

int status;

boolean finished;

};

This structure will be malloc'ed at the start of the game.

* The playerNames[0] will store the name of the first player and playerNames[1] will store the name of the second player. This is limited to 50 characters. We include an #define for this

#define MAX\_NAME\_LEN 50

 The status stores the state of the game. The game can be in 5 different states

P1\_TURN – it is player 1's turn

P2\_TURN – it is player 2's turn

P1\_WON – player 1 has won

P2\_WON – player 2 has won

DRAW– it’s a draw

We can an use an enum to store a symbol for each of these states. This makes the code easier to read.

enum status { P1\_TURN, P2\_TURN, P1\_WON, P2\_WON, DRAW };

The board is a set of 3\* 3 characters that can be one of three values. 'X' , 'O' or '-'

We'll include three constant characters for each of these.

const char SPACE= '-';

const char X\_SYMBOL = 'X';

const char O\_SYMBOL = 'O';

* Each time the player chooses a slot in the range 0 to 8 this is mapped to an element in the board array and the correct symbol is store in the board array.

When the user choose 0 the value of board[0][0] is updated

When the user choose 1 the value of board[0][1] is updated

When the user choose 2 the value of board[0][2]is updated

When the user choose 3 the value of board[1][0]is updated

**col 0 1 3**

**row 0** 0 | 1 | 2

----- ----- -----

**row 1** 3 | 4 | 5

----- ----- -----

**row 2** 6 | 7 | 8

The row and column can be calculated for each slot number using % an /

col = position % 3

row = position / 3

When the row and column are identified we check that the current symbol in the array is '-' which indicates the slot is free. The slot is then updated with the correct symbol for the current player. If the slot is not free the player is given another chance to enter a number.

# Version 1.

This version will display the initial board which is empty.

It will display the numbers of the slots and the first player can takes a go where he specifies a slot in the range 0 - 8

The board is updated to show the move.

# Version 2.

Add a game loop so that each player has alternate goes until the game is finished.

This version will identify if a player picks a slot that has already been played.

This version will identify if all the slots have been played and then set finished to True and announce a draw.

# Version 3.

This will identify when a player has won and which player has won and set the status and finished field accordingly

This requires traversing the board looking for 'wins'.

We know that if there are all X's in **row** 1 that player 1 has won. This **row** consists of

slots 4, 5 and 6, or these may be identified with rows and columns (1,0 ) ( 1,1) and (1,2).

To spot this winning avenue we would need to check that the values stored in board[1][0] , board[1][1] and board [1][2] are all the same and not '-' .

If the board[1][0] board[1][1] and board[1][2] have 'X' then player1 is a winner.

If the board[1][0] board[1][1] and board[1][2] have 'O' then player2 is a winner.

To remember these board positions we can store the six numbers above {1,0,1,1,1,2}in an array. There are 8 possible winning avenues of which this is one. We can use an 8\* 6 integer array to store all 8 solutions.

Can you identify the other sets of numbers that will need to be stored to help identify all possible winning avenues?

**Due Date:**

End of Week 11.

**Submission Details:**

To be added see blackboard.

**Lab Demo:**

A brief individual interview about the assignment will take place on our lab session the week 12. The Marks are based on the demo and by the lecturer reviewing and testing the code.