The assignment

For assignment four, you will write a Mancala server. Players will connect using *nc*. The `-p` option specifies a port number to listen on, which otherwise defaults to 3000. Use the supplied starter code to parse the command-line options. You may assume that if a port number is provided, it will be an integer.

To avoid port conflicts when testing your programs on teach.cs, use the `-p` option to specify a port number and select that number using the same algorithm we used for lab 10: take the last four digits of your student number, and add a 5 in front. For example, if your student number is 1008123456, your port would be 53456. Using this base port, you may add 1 to it as necessary in order to use new ports (for example, the fictitious student here could also use 53457, 53458, 53459, 53460). Sometimes, when you shutdown your server (e.g. to compile and run it again), the OS will not release the old port immediately, so you may have to cycle through ports a bit.

When a player connects, the server will send them "Welcome to Mancala. What is your name?". After they input an acceptable name that is not equal to any existing player's name and is not the empty string, they are added to the game, and all existing players (if any) are alerted to the addition.

With your server, players can join or leave the game at any time. A player joining the game has their pits initialized as follows. If they are the first player (or all other players have quit), they get the usual four pebbles per non-end pit. Otherwise, each non-end pit gets the average number of pebbles of all current players' non-end pits, rounded up. (See `compute_average_pebbles()` in the starter code.).

You will store the list of connected players as a linked list. New players are added at the top of the linked list. (It's easiest to implement this way.) Adding a new player must not change whose turn it is, unless this is the first player to join the game. Also, when someone leaves, their board is removed from the circle of boards, along with the pebbles in their board at that time. If it is their turn, the turn has to pass to the next player in the circle, not to the top of the list.

For each turn, or to newcomers joining the game, you display the game state in a simple text format. The display looks like this:

```
    mwc:  [0]5 [1]1 [2]6 [3]7 [4]6 [5]5  [end pit]0
    jpc:  [0]4 [1]4 [2]0 [3]5 [4]5 [5]0  [end pit]2
```

That is, one user per line, saying the number of pebbles in each pit (e.g., mwc's are 5, 1, 6, 7, 6, and 5, respectively), and identifying the pits by index numbers (starting from zero) for the use of players in making their moves.
Then, prompt the player whose turn it is with the query "Your move?", and tell everyone else (for example) "It is mwc's move."

The player is expected to type a single digit, followed by the network newline. Be permissive about other newline conventions; In other words accept either \r or \n as indicating a newline. Also, ignore all spaces (e.g., the user might type space, 3, newline; and this is a valid way to specify pit #3). The focus of this assignment is not error checking and so for the purposes of A4, you may assume that when it is their move, players always enter a number (e.g., the player will never enter z or 4a or b52). You may not assume that the number is a valid pit and if it isn't you should tell the player this and ask them again for their move.

Then tell everyone what move that player made. (The exact format of this message is up to you.)

Be prepared for the possibility that the player drops the connection when it is their move. Furthermore, you must notice user input or dropped connections even when it isn't that player's move. If the player types something other than a blank line when it is not their move, tell them "It is not your move." For a blank line, you can say "It is not your move" or you can ignore it, whichever makes your code easier.

As players connect, disconnect, enter their names, and make moves, the server should send to `stdout` a brief statement of each activity. (Again the format of these activity statements is up to you.) Remember to use the network newline in your network communication throughout, but not in messages printed on `stdout`.

You are allowed to assume that the user does not "type ahead" -- if you receive multiple lines in a single read(), for this assignment you do not need to do what is usually required in working with sockets of storing the excess data for a future input.

However, you can't assume that you get the entire player name in a single read(). For the input of the player name only, if the data you read does not include a newline, you must loop to get the rest of the name.

Submission

First, login to MarkUs to ensure that the proper subdirectory gets created in your repository and to get the starter code.

Commit `mancsrv.c`. No other files will be accepted. Please commit to your repository often, so that you don't lose your work.

Remember to also test your code on teach.cs before your final submission. Your program must compile cleanly (without any warning or error messages) on teach.cs (using flags `-Wall -std=gnu99 -g`) and may not crash (seg fault, bus error, abort trap, etc.) on our tests. Programs that do not compile or which crash will be assigned a 0.