CSC B09 Assignment 4, Winter 2018: Mancala

Due by the end of Friday April 6, 2018; no late assignments without written explanation.

The game of *Mancala* is played with pebbles and a board with indentations (depressions). This is a two-player board:

(image credit: Colin M.L. Burnett; see https://commons.wikimedia.org/wiki/File:Wooden_Mancala_board.jpg)

Four-player boards also exist. Through the magic of computers, your program for this assignment will allow any number of players. (Even ridiculously high numbers.)

Each player has a side of the board. As you can see above, there are six depressions per side plus a larger depression at each end. Each depression is called a "pit". The larger depression at the right end of your side of the board is your "end" pit, and you want to get as many pebbles into it as you can.

Each player begins with four pebbles in each regular pit, and an empty end pit.

On your turn, you choose any non-empty pit on your side of the board (not including the end pit), and pick up all of the pebbles from that pit and distribute them to the right: one pebble in the next pit to the right, another pebble into the next pit to the right after that; and so on until you've distributed all of them. You might manage to put a pebble into your end pit. If you go beyond your end pit, that's fine, you put pebbles into other player's pits. However, you skip other player's end pits.

If you end your turn by putting a pebble into your own end pit (i.e. if it works out exactly), then you get another turn. There is no limit to how many consecutive turns you can get by this method.

After that, it's another player's turn. The game ends when any one player's side is empty. At the end of the game, each player's score is all of the pebbles remaining on their side (which will consist mostly of the end pit, and will consist exclusively of the end pit for the player who emptied their side).

(If you end your turn by putting a pebble into a non-end pit on your own side which was formerly empty, there's another rule which gets you some of your opponent's pebbles at that point, but this is trickier to generalize to an arbitrary number of players, and doesn't add anything as far as this assignment is concerned, so we're going to ignore that rule for this assignment.)

You can learn your way around this game further by playing it at `http://play-mancala.com`

The assignment

For assignment four, you will write a Mancala server. Players will connect using *nc* or similar. The '–p' option specifies a port number to listen on, which otherwise defaults to 3000. Please use the supplied starter code to parse the command-line options.

When a player connects, the server will send them "Welcome to Mancala. What is your name?". After they input an acceptable name which 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).

(over)

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, if it is their turn, the turn has to pass to the next player, 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:

```
ajr:  [0]5 [1]1 [2]6 [3]7 [4]6 [5]5  [end pit]0
eem:  [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. mine 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 ajr's move."

The user is expected to type a single digit, followed by the network newline. Be permissive about other newline conventions; one way to implement this is to accept either \r or \n as indicating a newline, and to ignore blank lines. Also, ignore all spaces (e.g. the user might type space, 3, newline; and this is a valid way to specify pit #3).

Then tell everyone what move that player made.

Be prepared for the possibility that the user drops the connection when it is their move. Furthermore, you must notice user input or dropped connections even when it isn't that user's move. If the user 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 simpler.)

As players connect, disconnect, enter their names, and make moves, output a brief statement of each activity to stdout.

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. Please see testing notes in the Q&A web page.

Please see supplied starter code in `/cmsfaculty/ajr/b09/a4/starter.c`; you don't actually have to implement quite all of the above.

Other notes

Call your program "mancsrv.c". Your program must be in standard C. It must compile on the UTSC linux machines with "gcc –Wall mancsrv.c" with no errors or warning messages, and may not use linux-specific or GNU-specific features.

Submit in the usual way, with commands such as

```
submit -c cscb09w18 -a a4 mancsrv.c
```

Please experiment with a sample solution (compiled, so that you can't see the source code before the due date!), in `/cmsfaculty/ajr/b09/a4/mancsrv`

Please see the assignment Q&A web page at

```
https://mathlab.utsc.utoronto.ca/courses/cscb09w18/a4/qna.html
```

for other reminders, and answers to common questions.

Remember that some of the program is supplied for you as "starter code".