CSC 209 Assignment 4, Summer 2017: Dance Card Server

Due by the end of Sunday August 13, 2017; no late assignments without written explanation.

This assignment involves completing the code for a server which emulates the partnering of dancers for a couples-oriented dance. In some forms of dance, people dance in twos and there is etiquette around asking someone else to dance.

Often there are two distinct dance roles for the two people in a couple. These roles can be generically referred to as “lead” and “follow”. Some dancers may be able to dance only one of these two roles; others may be comfortable dancing either role.

There is an already-implemented client for this client-server protocol, called “dancer”. The source code is in /u/csc209h/summer/pub/a4/dancer.c. Each dancer would run one of these clients, which would connect to a “dancesvr”, which you will complete the implementation of. There is a simple protocol defined in /u/csc209h/summer/pub/a4/PROTOCOL; mostly, the connection begins with a banner, and the transmission of the dancer’s name (“handle”) as well as whether they dance lead, follow, or are able to dance both roles. (Really this should be called “either” role—you can’t dance both roles at once!—but the usual brief word is “both”.)

The server disallows duplicate handles, or the handles “who” or “begin” or “debug”.

After a new dancer connects and completes the initial data entry, the server tells everyone else that that dancer has entered the dance hall. Dancers can type “who” to see a list of available dancers. You can make “debug” do whatever you want for your own purposes, if you want; or you don’t have to treat it specially if you have no use for it.

You ask someone to dance by typing their name. The etiquette around declining a dance invitation varies between organizations, but we will assume for simplicity that the person being asked always accepts. However, you can’t make the mismatch between leads and follows worse by whom you ask to dance. For example, consider a dance hall in which there are four people: Laura only dances lead (her name begins with ‘L’); Fiona and Ferdinand only dance follow (‘F’); and Beth can dance either role (remember ‘B’ for “both”). In this setup, Laura and Beth will not be permitted to dance together, even though they are able to, as follows. If Beth dances with Fiona or with Ferdinand, then everyone can dance (the other one of Fiona and Ferdinand can dance with Laura), whereas if Laura and Beth take each other as dance partners, Fiona and Ferdinand are going to have to sit out the dance because they can’t dance with each other because they are each only able to dance follow. Therefore Laura and Beth aren’t allowed to take each other as dance partners.

The formula for determining how many people are necessarily left out of the dance, given a certain number of remaining unpartnered leads, follows, and “boths”, is most easily cast in pseudocode rather than English:

```plaintext
x = max(nlead, nfollow) - min(nlead, nfollow) - nboth
if (x < 0)
    return 0
else
    return x
```

Also note that `max(nlead, nfollow) - min(nlead, nfollow) = abs(nlead - nfollow)`.

So, if this value is zero, two people are prohibited from dancing together if they would make the value positive (non-zero). If it is non-zero, two people are prohibited from dancing together if this would increase the value (which is just a generalization of the previous sentence).

After everyone is paired, or no more pairs are possible, the dance begins. Alternatively, anyone can cause the dance to begin by typing “begin”. The server sends a suitable message, does sleep(5) (it’s a very quick dance), then sends a message about the dance ending.

When the dance ends, everyone is unpartnered, and can choose partners for the next dance.

Do not begin the dance automatically when there are no partnerships. For example, when the server first starts up, there are zero dancers unpartnered, but the dance should not immediately begin!

(continued)
Data structures

The supplied starter code in /u/csc209h/summer/pub/a4/dancesvr-starter.c contains a “struct dancer” to represent each connected client; you will maintain a linked list of these objects, and you can add additional fields to the struct as needed.

You will want to do something like:

```c
#define FOLLOW 0
#define LEAD 1
#define BOTH 2
```

This gives you the ability to make comparisons like:

```c
role == FOLLOW: can only dance follow
role != LEAD: can dance follow (perhaps can dance both)
```

and if you have two people with roles role1 and role2 respectively, you can say

```c
if (role1 != role2 || role1 == BOTH)
```

to check whether they can dance with each other. (This is a mathematical simplification of the more obvious expression `(role1 == BOTH || role2 == BOTH || role1 != role2)`.)

Your server will keep integer totals of how many unpaired people are in each of the three categories: follows, leads, boths. After a dance, go through the linked list of clients to regenerate those values.

The supplied struct dancer also contains a field “partner” to indicate who this dancer’s partner is. It should be NULL when first connected or after a dance ends. You will also have to reset it to NULL if the dancer’s partner quits the client program before the dance begins.

Sample messages to send to the client can be obtained by trying my compiled solution in /u/csc209h/summer/pub/dancesvr, as discussed in lab 12. You do not need to use any of the sample messages, or any of the starter code, but I suggest doing so unless you have a reason to deviate.

Suggested sequence of implementation

1. Do lab 12, in which you “play client” and “play server” to understand the protocol.

2. Run my client and server in several windows (one server, several clients) and try different combinations of dance roles and asking people to dance and so forth—just play with the program a bit to understand what it does.

3. Copy /u/csc209h/summer/pub/a4/dancesvr-starter.c to your own directory and rename it to “dancesvr.c”. Look through the code and understand its organization. You have a lot of code to add to this, but many of the trickier bits are supplied for you. (You also get to write some of the tricky bits.)

4. Write newclient() and removeclient() — code to add and remove clients from the global linked list of struct dancer objects. Think carefully about how to initialize the various members of the struct in newclient(), and what other cleanup is needed in removeclient(). The newclient() function also sends the “dancecard 1” banner.
5. Write select()-using code in main() or in a separate function called from the appropriate place in main(). When there is read data available from an existing client, call clientactivity(); when there is a new connection call accept() and newclient(). Also write the calls to bind and listen in makelistener() (or however you want to organize this).

6. Add an appropriate read() at the beginning of clientactivity(). At this point you should have something you can run, although it’s not going to do a lot, and the client will hang after soliciting the user’s handle because it won’t get a happy response from the server. However, the clients should connect and prompt for the handle at least (showing that they’ve verified the banner).

7. Don’t proceed until you have the above working, including that your program compiles with “gcc –Wall” with no warnings. From now on in this assignment, you should always have a working, warning-free program. After completing each step, I suggest making a new backup copy of your dancesvr.c, perhaps named with the item number which it represents successful completion of.

8. Do what’s needed to allow clients to set their handles, to choose their roles, and to ask someone else to dance or to type a command (even though the asking someone else to dance or typing a command won’t work yet). The messages such as “Beth has joined the dance” should be sent to clients as applicable. Remember to prevent duplicate handles, or the handles “who” or “begin” or “debug”.

9. Implement the ‘who’ command, which also runs automatically for newcomers. Note how my supplied solution indicates when prospective partners are incompatible (e.g. if both you and they can only dance lead).

10. Implement the choosing of partners, without any restrictions on who can dance with whom.

11. Clean up properly if a partnered dancer quits. The partner should be notified and should be able to choose a new partner.

12. Implement the restrictions on the choosing of partners, in terms of who is compatible with whom, and in terms of not choosing a partner in a way which will leave more dancers out.

13. Add code to begin the dance automatically when applicable.

**Other notes**

Your programs must be in standard C. They must compile on the teach.cs machines with “gcc –Wall” 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 csc209h -a a4 dancesvr.c
```

Please see the assignment Q&A web page at


for other reminders, and answers to common questions.