

## CSC209H Worksheet: Sockets

In this worksheet, you will write a tiny interactive game for two players who will connect over the internet. Your instructor will demonstrate the game at the beginning of lecture.

When completing the steps below, look for the `TODO` comments throughout the code that correspond to various steps.

1. Download the `lego_game.c` starter code from the course website.
2. If you are working while SSH'd into a lab PC, change the port number in the code from 54321 to a number that hopefully will be unique to you and avoid conflicts with other students doing this worksheet. To create your port number 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.
3. Compile and run the example program before making any other changes. From another terminal (maybe even another lab PC!), connect to your server using the command:  
`nc -C dh20XXpcYY.utm.utoronto.ca <54321> // replacing <54321> with your custom port number.`

If you ran the program locally, connect to your server using the command:

```
nc -C localhost <54321> // replacing <54321> with your custom port number.
```

Note: we are using the `-C` flag to send `CRLF` as the line-ending. On some machines, that option is `-c` (lowercase), instead of `-C` (uppercase). For more information, consult the `man` page for `nc`.

4. Now it is time to change this starter code into a server that will play the chocolate bar lego game. Start by adding the ability for a second client to connect. Change the code so that when the first client connects, they are told that they are player 1 and that we are waiting for player 2. Then call `accept` again and wait for a second connection. Once you have two clients connected, send them both a message to tell them the basic rules of the game. Compile and try running your code.
5. You are going to send quite a few messages to both players. Create a helper function `write_to_players` that takes a string and the two socket descriptors, and writes the string to the sockets for both clients.
6. Next write a helper function `read_a_move` that returns a valid integer representing a single move. There are a number of design choices here and you have some flexibility. One option is to take a socket descriptor as a parameter and **repeatedly** read and convert the response from that descriptor until the player enters a valid move (a move between 1 and 3.) Remember that the player is going to send the integer as text followed by a network newline. Before you can call `strtol` to convert this response to an integer, you will need to ensure that it is actually a legal string.
7. Test and debug your `read_a_move` function by calling it with one player or the other before trying to code the context of the game. For the purposes of this worksheet, assume your players are “good citizens” and not trying to break your server (for the tutorials and assignment, your client/server should also handle cases where a hostile user is actively trying to cause your client/server to misbehave or crash). You should make sure though that if a player asks to take fewer than 1 items or more than 3, you tell the player that this isn't allowed and ask for the move again.
8. Once you are convinced that you can read moves from the players, fix up the actual game logic (look for the `FIXME` strings in the code). You will need to strictly alternate turns and between each turn, tell both players how many blocks are left before the chocolate bar. Once the game is over, your program should say who actually ate the chocolate bar and then exit.
9. There are *many* places where you may have cut corners on good program design. If you have extra time, go back and fix them now.