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. If you are starting this worksheet before the class has started, you can complete steps 1 through 4 even before understanding the game.

1. Download the example code `server2.c` from the sockets videos on PCRS.

2. 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. Add the following code to your program between where the `listen_sock` is declared but before it is bound to an address. This sets an option on the socket so that its port can be reused right away. Since you are likely to run, stop, edit, compile and rerun your server fairly quickly, this will mean you can reuse the same port.

```
int on = 1;
int status = setsockopt(listen_soc, SOL_SOCKET, SO_REUSEADDR,
                        (const char *) &on, sizeof(on));
if(status == -1) {
    perror("setsockopt -- REUSEADDR");
}
```

4. Compile and run the example program before making any other changes. From another terminal (maybe even another machine!), connect to your server using the command
`nc -c wolf.teach.cs.toronto.edu <54321>` // replacing `<54321>` with your custom port number.

5. Now it is time to change this sample code into a server that will play the chocolate bar 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.

6. You are going to send quite a few messages to both players. I found it helpful to do two things:

   (a) Create a helper function that takes a string and writes it to the sockets for both clients.

   (b) Make the two socket descriptors global variables (so you don't have to pass them to this helper function every time.) This is completely optional, but makes your code shorter and easier to read.

7. 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. So before you can call `strtol` to convert this response to an integer, you will need to ensure that it is actually a legal string.

8. 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. Don't get too hung up on the corner cases. For the purposes of this worksheet, assume your players are "good citizens" and not trying to break your server. 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.

9. Once you are convinced that you can read moves from the players, code up the actual game logic. 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.

10. 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.