**Create a sign.**

You will be writing a program that implements a toy LED sign (we will call this Program A) and a separate program (Program B) to tell the sign program what to display. You will then be parsing “bus locations” with Program B and updating the sign when “stops” are detected. Feel free to use libraries.

1. Program A --

Write a command line “server” program to visually display a 7x128 “pixel” sign. At any moment, each pixel is either “on” or “off”, with “on” represented by “#” and “off” represented by a space.



letters.txt contains some banner-style letters that might be useful. The first line of letters.txt is a string containing the characters in the order that they appear in the rest of the document. The banner characters are separated by a single blank line. Note that the letters are not fixed-width. Some characters, such as double quotes, are not represented.

As an alternative, feel free to implement a sign differently, perhaps using libraries or frameworks. The CLI approach is only a suggestion (we are more interested in what you do with the sign than in how it is implemented).

Your sign program should be long-running and able to update the display based on a signal from another program.

1. Program B --

Write a 2nd program to tell the first program what to display on the sign. The type of signal used is up to you -- for instance, the signal can be over a network or via IPC.   
  
What happens if the text does not fit on the sign? What happens if new display text is sent while the sign is still in the middle of displaying a previous message (perhaps the previous message was very long)? Does the 2nd program ever need to query the 1rst program or send any commands besides the text to display? This is up to you.

This 2nd program should be able to be executed as a one-off CLI executable that transmits the command line arguments. However, this is not the only way that this program is used...

1. You have been provided with locations.csv and stops.json. The locations file contains a time series of location information from a bus that was driving a route, and the stops file contains a list of bus stops on that route and some associated information such as the stop geofence (lat, lon, and radius in meters) and stop name. Some of those stop names are curiously long!  
     
   Read 1 line of locations.csv every second and check to see if the bus is now in a stop (hint -- look into haversine calculation. Feel free to use libraries). If it is, update the sign!
   1. When a bus gets to a stop, visually display that stop.
   2. When the bus leaves a stop, display a “next stop” message.
   3. When the bus is not at a stop, alternate between displaying the route name and the current time.

Clearly locations.csv is just for testing -- real buses get their locations from a GPS source, not a text document. Keep this in mind when implementing the code that reads locations -- we should be able to switch the implementation out when we are ready to go to production! Don’t worry, this isn’t actually going into production. ;)

Optional -- add the ability for the 1st program to scroll, and leverage this in the 2nd program. What new design challenges that arise from scrolling text? Should all messages scroll, or just certain messages?

Optional -- We just received a new sign that is 15x196 pixels and allows for 2 rows of text! Implement this.

Optional -- We received a new sign that is 64x64, and a client has requested that we implement remote 2-player pong (a frequent request from larger transit agencies). Have at!

Conversation topic -- Customer support is concerned about interior sign support calls and wants to be able to remotely see what is being displayed on the sign. How can this be accomplished?

Provide sufficient documentation for this to run on our machine(s).