Final Project for 07-120

In the final project for this class, we will make widgets for the custom watch we've given you. The widget can display a summarized or visual representation of any data you want. For example, weather, stocks, live scores from your favorite sports, your physical activity data, etc.

You will use the GitHub repository invite for all of your final project code, script(s), diagram(s), and reflection writeup.

Given that we cannot yet program the watch itself, you will do all the processing locally on your computer and push the contents to the watch over a USB connection.

The watch contents can be controlled using a series commands from the computer. There are several components to the pipeline here:

1. **The Watch**: It runs an Arduino-based firmware that we do not need to touch for this project.
2. **Processing IDE**: We use Processing to establish a Serial communication between the watch and computer. The given Processing code reads the content of a file on your computer and sends the contents, line by line, as a string to the serial port.

   You must ensure that the code talks to the correct serial port. The file from which data is sent to the serial port needs to follow the commands that the watch firmware understands.
3. **C0 executable**: This is one of your main tasks in this project. You will do data parsing and wrangling in your C0 code to extract meaningful information from the data provided by some data source and ensure that the processed data is something the firmware on the watch understands and can display clearly.

   In addition to the code itself, **You should provide tests and contracts for your C0 code.** You can also do this part in C, if you so choose.
4. **Shell script**: Given that we cannot query the Internet using C0, we will rely on the Shell command, `curl`, to query the websites that will provide our data of interest. We will also use a Shell script to transform the data from the Internet into a format we can parse in C0. We encourage you to convert any acquired data into *CSV format* before reading it with your C0 program. Finally, you can use Shell scripts to regularly update the contents of the watch by running your pipeline repeatedly. **Configure the script to run your code once every two minutes**.

Watch Commands

There are two main types of commands you can run:

1. `<clear>`

This command clears the contents on the screen and displays the time on the watch (ignore the time it shows as it can often be wrong or run too fast).

2. `<position;color;type;value>`

This is the main command that changes the contents on the watch display. It has four semi-colon-separated fields:

- `position`: As the name suggests, this field is where the new content will be placed. It has three possible values: *top*, *middle*, *bottom*.
- `color`: This field specifies the color of the new content. Possible values include: *black*, *brown*, *red*, *orange*, *yellow*, *green*, *blue*, *purple*, *grey*, *white*, *cyan*, *magenta*, *pink*.
- `type`: The watch supports four types of content: `string`, `number`, `range`, and `chart`.
- `value`: For each type of content, the values will differ:
  - `string`: It can be any string up to the length of 15.
  - `number`: It can be any number up to 99,999.
  - `range`: It is a pair of numbers (size up to 9,999). Separate the two numbers by a comma (`,`).
  - `chart`: This is a collection of 20 comma-separated values. Keep the values between 0 and 10. If your data values are larger than 10, you must scale everything down to be between 0 and 10. Larger (or smaller) values will show up funny on the display.

Processing IDE Application

We have given you a program titled `read_file_and_write_to_serial.pde` in your repository. This code reads from a file called `commands.txt` and outputs its contents line by line to the serial port. You can change the program to read the contents of another file, but it is not needed as long as your C0 and Shell script pushes your parsed content to `commands.txt`. When the Processing application runs, it searches for all open serial ports and prints them to the console inside the Processing IDE. Browse the list to find the port that is connected to the watch. It should show something similar to `/dev/cu.usbmodem1101`. If there are multiple ones, you should experiment to find the correct one. The program's output lists the indexes on the left of the serial port. Update to the corrected index on line 13 of the Processing code.

As an example, for the code snippet in class, we used the index 3:

```
myPort = new Serial(this, Serial.list()[3], 9600);
```

Make sure this program is running at all times when you are looking to interact with the watch.

Expectations for the Final Project Deliverables

1. **Choose a data source** that can provide data from an online API (typically called using `curl` command on the shell) or that could be manually downloaded from some source data, e.g. website, physical activity data from a phone if you intend to mimic FitBit-esque App, etc.

   Please share your data source(s) in the #general Slack channel.

2. **Diagram your implementation** of the project, similar to what you did for Wordle in homework 4. Draw out the steps/flow of your implementation and how components and functions connect. If you remember from class, this should detail how you gather and export data, how you plan to parse it into the necessary format, and how you plan to display it.

   You can commit an image, pdf, etc to your repository for review.

3. **Implement at least two widgets** for an application of your choice that includes **at least one graph output** that involves some summarization or processing of the data parsed from an external source. We expect that your repository will have C0 (or C) code and Shell script(s).

   An an example, you could have one widget that only shows the temperature number and the city, and then another with a view which shows city, day range, and a graph for that day.

   Your final C0 (or C code) must include tests and contracts, covering most of the necessary inputs, outputs, and logic.

4. **Provide documentation** for how we can run your project. This includes what script(s) to run and any other helpful information. It should cover everything needed to run your project. You can use the provided `Readme.txt`.

5. **Reflection**. We're looking for a reflection on your process in designing, implementing, and debugging this project.

   - *Q1*: Where did you get stuck and how did you debug your problem(s)?
   - *Q2*: What commands were most useful?
   - *Q3*: Did you do any refactoring along the way?
   - *Q4*: What concepts from the class ended up being the most helpful to you?

Deadlines and Deliverables

1. Provide diagram(s), data source(s), and an *Hello World* display example.

   Due Friday, December 1, 2023 at 11:59 pm.

   - **Sources should be provided and shared on Slack for everyone to use**. We do expect at least one suggestion per individual.

- **Diagram(s) should be pushed up to your repository** (PR'ed and merged).
- Using the given Processing program, have the watch display *Hello World*.

  Upload a screenshot, gif, or video (better) demonstrating this working to your repository.

2. Give an in-class presentation with a basic working version of the project.

   Due Wednesday, December 6 2023 at 11:00 am in class.

   - In class, you'll present your in-progress project, which should demonstrate at least one working widget with the watch, even if it's in rudimentary form.
   - Prepare one slide discussing the questions centered around your reflection.

     Please send us your one slide ahead of class by DM on Slack.

3. Merge your final, working code (with tests and contracts) for *at least two widgets* of your application. Include instructions/documentation for how to run your project, and provide a written reflection.

   Due Friday, December 15 2023 at 11:59 pm.

Grading

The total assignment is worth 180 points and is 40% of your final grade.

Item	Points
Implementation Diagram(s)	20
Provide Data Source(s)	10
Hello World Display on Watch	20
Work In-Progress Presentation	50
Final Code (with tests, contracts, 2 widgets)	45
How-to-run-it Instructions/Documentation	10
Reflection	25
Total	**180**