PENN ELECTRIC RACING

SOFTWARE CHALLENGE

Overview

Welcome to the Software Team Challenge! The goals of this challenge are for you to show us what you know, meet the team, and learn more about the car. The tasks listed below are designed to give you an introduction to what software team members are expected to do, so please read the tasks carefully.

We are not looking for a "correct" answer necessarily, more so looking for your effort, creativity, and desire to learn. This challenge is **difficult**. If you get stuck somewhere, email us questions, come to our office hours, and Google things. The team develops primarily in C++, but feel free to complete the challenge **with whatever language you want**. We would like to see everyone attempt all three challenges; if you get stuck on a part we would recommend moving on and coming back to it later.

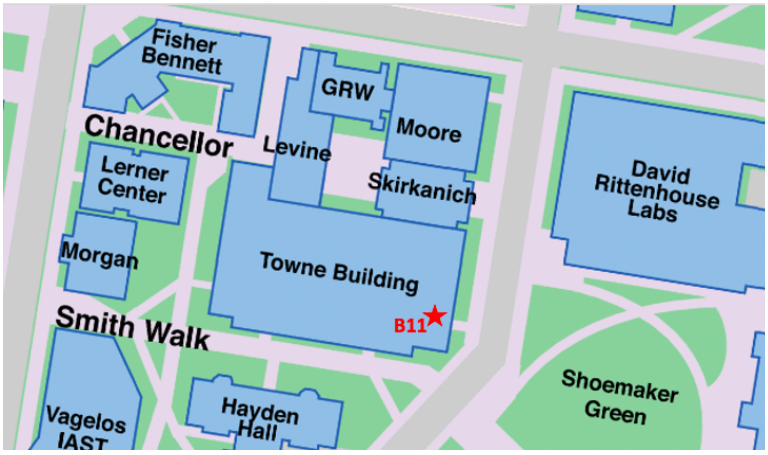
Should you have any questions, email the software lead Eric (ecao22@seas.upenn.edu), or stop by B11 (found in Towne Basement closer to the 33rd St side) during office hours (detailed below).

Please submit your completed challenge as a zip archive by **12:00pm EST (Noon) on Saturday, September 7th** to Dropbox: https://www.dropbox.com/request/pCwWUSK3poOLVo1sFxv0. The file name should be: "[LastName]_[FirstName]_SoftwareChallengeSubmission.zip."

Include the following:

- ***ALL*** of your code (don't delete things, just comment them out, we want to see the journey) and anything needed to compile and run it. ***Note:*** we mostly use Linux and macOS on the team, but we also have access to Windows machines.
- A README file, with the items that we describe at the end of this document.
- Other documents listed in the application email.
- Anything else that you think is relevant or interesting.
- ***DO NOT*** include any of the files that we give you (some of them are big).

Office Hours: Throughout the week we will have office hours in B11 where you can come and ask questions about the application or the team in general. **Office hours are September 3rd, 4th, and 5th from 5pm to 8pm.** If you cannot make these office hours, you can email the Software Lead (email above) with questions. You are **encouraged** to come to us with any/all questions you may have.



PENN ELECTRIC RACING

Software Challenge

Background

Our car has several custom circuit boards on it with microcontrollers, which are small and limited computer chips that run code and perform various functions for the car. These boards communicate with each other using the CAN (Controller Area Network) communication protocol. CAN allows any microcontroller to broadcast data packets to all other microcontrollers (or other CAN devices) on the car. These packets are sent at around 10-100 Hz (times per second), along with a unique ID to the message type and 0 to 8 bytes of data. The data is sent to a Data Acquisition (DAQ) board and is written to a log file on an SD card.

For the next three parts, you can download files from this shared folder:

https://drive.google.com/drive/folders/1VnoS8CPUe_9870Pjky-eK_8d_gvqfhxV?usp=sharing

There are two files with names corresponding to the parts they should be used for.

Tasks A and B Introduction

For this challenge, we're going to give you a few of our log files in custom file formats. You will be asked to parse these files, extract data from them, and perform calculations. For each part, first determine the **minimum**, **maximum**, and **average speed**, in MPH, of the car during the race. Then, calculate the amount of **power consumed** at each point in time using the current and voltage, and finally use numerical integration (another resource) to find the **total energy consumed** during the run. Please convert your units to kWh for consistency. For reference, our battery pack was approximately 5.2kWh during these runs and was designed to just make it through the longest FSAE event, Endurance. Keep in mind that these files contain real data from real car runs.

Task A: The CSV-ish File Format

The first file that we will give you is similar to a CSV (Comma Separated Value) file. It is from a past practice Endurance course run done in May 2023. Note that this is a text-based log format, and therefore, the best way to inspect the file is with a text editor. The file format contains three different types of lines of text:

1. The first line is a header, containing the following:

   `PER CSV Modbus Log` *`TIMESTAMP`*

   Where `TIMESTAMP` follows the format MM/DD/YY hh:mm:ss (AM|PM). An example:

   `PER CSV Modbus Log 06/18/16 01:29:12 PM`

2. A sequence of Value-ID associations of the form:

   `Value` *`VARIABLE_NAME (VARIABLE_DESCRIPTOR)`*`: ID`

In this case *VARIABLE_NAME* is the name of the variable measured on the car, *ID* is an integer representing its CAN ID, and *VARIABLE_DESCRIPTOR* is the C++ identifier for the variable. For instance:

```
Value Voltage (ams.pack.voltage): 8754
```

specifies that the `Voltage` (the voltage of our custom battery pack) will be referred to with ID 8754.

3. After the Value-ID associations comes the actual data in the log file. This is of the form

```
TIMESTAMP,ID,Value
```

where `TIMESTAMP` is in milliseconds from the start of the file and `ID` refers to the Value-ID associations from part (2). `Value` is a string-formatted version of the variable data. For Task A, you will only be dealing with floating-point values, but these values can be a few different datatypes.

Here are the relevant values that you will need to work with to calculate the metrics mentioned in the introduction:

Name/Descriptor	Unit	Description
`Voltage (ams.pack.voltage)`	Volt	The voltage of the accumulator (battery) during the run
`Current (ams.pack.current)`	Amp	The current of the accumulator (battery) during the run
`Front Left Wheel Speed (pcm.wheelSpeeds.frontLeft)`	MPH	The speed of the front left wheel
`Front Right Wheel Speed (pcm.wheelSpeeds.frontRight)`	MPH	The speed of the front right wheel
`Back Left Wheel Speed (pcm.wheelSpeeds.backLeft)`	MPH	The speed of the rear left wheel
`Back Right Wheel Speed (pcm.wheelSpeeds.frontLeft)`	MPH	The speed of the rear right wheel

Submission

For this part, provide the code you wrote to calculate each of the values, along with instructions on how to run it. In the README, include the minimum speed, maximum speed, average speed, in MPH, and total energy consumed, in kWh, as described above for the Task A file.

Task B: The Old Text Format

This format is also text based. It's data from our 2015 Endurance run, which we completed. The first line is a header, and after that, every line follows the same format:

```
TIMESTAMP CAN_ID BYTE_COUNT [ BYTE_1, …, BYTE_N]
```

Note that here `N` is equal to `BYTE_COUNT`. Here's an example:

```
12:38:43.518 2 0x22 8 [ 217, 184, 130, 65, 122, 175, 137, 65]
```

In the example above, the 8 bytes represent two different numbers as shown.

```
First number            217, 184, 130, 65
Second number                                 122, 175, 137, 65
```

The second and third values compose the CAN ID in hexadecimal. For instance, `2 0x22` corresponds to `0x222`, which is `546` in decimal (i.e. `3 0x12 = 0x312 = 786`). We then have the eight bytes associated with this packet of data (a packet of data can represent multiple variables). All of the values you are going to be looking at will either have 4 or 8 bytes of data, corresponding to one or two floating-point numbers, respectively. They are 32-bit little endian floating-point numbers. You'll need to figure out how to convert these bytes into floating-point values[1]. To check that you are converting correctly, the two numbers in the example above are `16.340258` and `17.210682`.

The data packets with the following CAN IDs are relevant:

CAN ID	Name	Unit	Description
0x311	Battery Current	Amp	The current of the accumulator (battery) during the run
0x313	Battery Voltage	Volt	The voltage of the accumulator (battery) during the run
0x222	Front Wheel Speed	RPM	The ***angular*** velocity of the front wheel. The first float in the pair is the left wheel velocity, and the second is the right wheel velocity.

Assume that the diameter of each wheel is 20.5 inches. This was before rear wheel logging, so approximate the vehicle speed using the data you do have.

Some data in this file may be corrupted or have edge cases. Use your best judgment on how to handle this, and justify your decisions in the README if you feel it's necessary.

Submission

[1] How you do this depends on which language you choose to use. In general, lower-level languages like C and C++ might make tasks like this easier, while adding complexity to the text parsing and processing.

For this part, provide the code you wrote to calculate each of the values, along with instructions on how to run it. In the README, include the minimum speed, maximum speed, average speed, in MPH, and total energy consumed, in kWh, as described above for the Task B file.

Task C: Custom Data Format

One of the most exciting aspects of working on Penn Electric's software stack is being able to architect an entire system: choosing the underlying technologies, languages, software structure, and support libraries that will be used. A critical part of the software design cycle is coming up with clear and coherent structure for how all these moving parts interact.

For this task you will be designing a data format for Penn Electric's embedded code. Each car we build has a predetermined set of variables of different types that get logged to an SD card on our on-board data logger. Your job will be to design a data format and write an encoder/decoder that can serialize this kind of data. Several design criteria you should consider:

1. **Extensibility.** The data logs you produce should be self-describing (i.e no external data needed to decode them) and be flexible enough to handle different configurations of variables. If you choose to base your data format after an existing one, the data format should be modifiable.
2. **Space efficiency.** Using a text-based format such as CSV is not space-efficient and logging CSV data on an embedded system is not possible when logging high-frequency data.
3. **Maintainability.** How debuggable is the resulting code? Is it easy to inspect your format with existing tools like hexdump?
4. **Portability.** How easy is it to port the format to another platform? Does your code handle issues like endianness on different systems? Would implementing the format for an embedded system (just generally in another language) take a lot of effort, or would it be straightforward?
5. **Corruption**. We do not expect you to handle data corruption issues in your format, but how might you go about making your file format resistant against missing or corrupted bytes?

Specifications

You may use **any** language to write your encoder/decoder.

Your data format should be able to losslessly represent the data described in the CSV-based logfile from Task A. See the Task A write-up for more information on the specifics of the file format, but there are a few new things to note. In Task A, you only interacted with floating-point values. In actuality, each of the underlying variables may be a boolean (encoded as 1 or 0), integer, or a float. Your data format **must** be able to handle all these different types (including 64-bit doubles/integers) and not only floats. You may assume that each variable (i.e. each unique ID) has a set datatype that is fixed for the entire run of the car and make data format choices appropriately.

You should carefully think about what tools you will be using and make sure to do a lot of googling on existing data encoding methods to get some inspiration. Your final solution, however, should be

your own implementation and **should not use existing standard libraries or external libraries for encoding data in a pre-existing file format**. Generic tools for encoding/decoding primitive values are fine, however. We are not expecting an industrial-grade software library or file format, but it should, at a minimum, outperform the CSV format we have provided space-wise.

Submission

For this part, please submit a short explanation of your data format and why you made the design decisions you did with respect to extensibility, space efficiency, maintainability, and portability. This should be included in your README (see below).

For the code submission, you should produce two executables: one that converts from CSV to your format and one that converts back.

README File

A README must be included with information on all 3 parts of the challenge. Feel free to use text, markdown, or just a regular PDF file. It might be useful to think about these and answer them as you or working. Note that only the first three bullet points are required.

Required Parts:

- The justification and results for all the parts you completed (you should put the justification and file format description for part C here).
- Clear instructions for how to compile and test your code for each part.
- A description of what you could accomplish and what you learned.

Optional Parts:

- What was the hardest part?

Tasks A & B

- What are some of the pros and cons of each log format? Why?
- How fast are your parser(s)? What could you do to improve performance? What's the time complexity of your algorithm?
- Can you create a graph of some of the data? Feel free to use Excel, Numbers, etc. You can graph whatever you want, but I suggest looking at the front and rear wheel speeds from an old acceleration run, which was that fourth shared file.
    - Do you notice anything interesting?
- Why might front and rear wheel speeds produce different results?
- Each piece of data has to be sent out separately, meaning that you could get two different pieces of information you need at different times. How did you deal with this? Is there a better way?

Task C

- How fast are your encoders/decoders? How could you improve some of the performance?