Project Breakdown

The project has 5 repositories that have been created as our scope shifted. These can each be found in our GitHub group here: https://github.com/COSC320-2023-Group-B
Most of the work did not contribute to the final product, and has been kept to showcase research, previous work, and in case snippets of code can be reused.

The first project, capstone, was created with the goal of connecting to the Caretaker from a Raspberry Pi and, shortly after, directly from a computer. It contains the caretaker library with an example project and some failed attempts to use the project (a makefile and a shell script). At this point we didn't understand how to use Visual Studio projects and although none of this code contributed to the final product, there are useful snippets here for potential future development if roadblocks can be overcome. The same goes for the second project caretaker-link, which attempts to import the library with Visual Studio 2022, but can't build due to default project settings that aren't compatible with the library.

The third project ctlink holds the most valuable caretaker linking work. After resolving the Visual Studio issues, a program was created that allows the user to connect to a Caretaker via a terminal interface, extract data from it, and produce a live-updated .csv file that would later be read by another program for LabChart. This unfortunately couldn't be tested, as the Bluetooth Low Energy connection that the Caretaker requires wasn't able to be made. The repository contains a main.cpp file under ./ctlink/main.cpp which utilises a set of functions from callback.h to handle events like a device connection or data being received. It also uses csv_creator.h to format data, create .csv files, and append to them.

The fourth project LightningDeviceSDK is a private fork of ADInstruments' LightningDeviceSDK repository. This was made to create a TypeScript plugin that would read from the .csv file and feed LabChart with data as it came in. After it became apparent it wasn't possible to connect to the Caretaker as planned, this project was deprecated before any major work was done.

The fifth and final project Converter contains the only complete and fully-functional software. It began with some C files in ./c-src that could take a .csv file and create a new one. Then we converted the project to Python, as Python has more convenient data manipulation functionality. The Python files include labchart_converter.py, which has a set of functions used by CSV_converter_GUI.py. CSV_converter_GUI.py acts as the program's entry point, and will create a TKinter window that allows users to select an input file, and specify how the output file is produced. When the user selects the convert button in the GUI, the labchart_converter.py function convert_CSV_To_Labchart will be called, which will step through five functions. First, it checks if a .csv file is a valid Caretaker data file by confirming that the headers are correct. Then load_csv is called, which will read the data file and generate a dictionary connecting headers to arrays of data. get_header_information is then called, and returns a list of headers to be used in generating the .csv file. Next, lerp_data is used to convert the irregularly sampled data points from the Caretaker into regularly spaced data points linearly interpolated from their position between the two nearest data points. Finally the headers and the data is brought together in a .csv file that can be read directly by LabCharts with save_to_file.