CS23820 Assignment 2024 – 2025
Fish Feeder Firmware

Neal Snooke

October 31, 2024

1 Introduction

This document describes the 2024-2025 assignment for module CS23820 and is worth 60% of the module assessment (around 40 hours work for the average student).

Where this assignment specification states any exact requirements, you MUST follow them precisely, in other areas you may take decisions as you see fit but you must document and justify those decisions. This assignment specification contains many details and you will need to read it very carefully and probably multiple times. The appendices contain technical information and examples you will need to consider during design and implementation.

2 ANSI C language and development environment

You must use ANSI C to implement this assignment. We recommend you use an IDE and debugger to assist in writing the code. CLion is the recommended IDE. C11 or C17 would be a good choice of version.

We strongly recommend that you ALWAYS specify the appropriate flags to the compiler to force it to give you warning or error messages if you stray outside of ANSI C. When assessing your work we may use various software to help us verify ANSI C compliance and program correctness and to detect unacceptable academic practice.

3 The problem to be solved

Your task is to *develop the software* to control a stand-alone electronic fish feeding device and *not* to design the hardware. The hardware is based on a low power microcontroller and includes a small OLED screen, press button and a stepper motor that rotates a drum to dispense pellet food. The device also contains a real-time clock chip and has no internet access. The hardware has been prototyped and tested with a simple program that displays the clock and rotates the drum when the button is pressed.

The software required will create a fully functioning programmable fish feeder with the following high level requirements:

- rotate the feeding mechanism for a specified number of rotations at the times specified by the feed schedule (see below). The feeding mechanism should have a maximum

rotation speed of one rotation per 20 seconds[1]. Note that the feeding process may therefore take some time.

- provide a 'main screen' that includes in the display: the current time; next feed time; number of feeds since the automatic feeding schedule started; feeder operating mode (see below); access to configuration menu(s);
- provide configuration menu or menus to allow the user to: set the clock; configure the feed schedule; select the operating mode as follows:

  paused no feeding occurs;

  auto feeding occurs according to the programmed feed schedule;

  feed now allows the user to initiate one rotation of the feeder mechanism (upon completion the feeder mode will return to the previous mode);

  skip next feed the feed schedule starts only after the next programmed feed time.

- all menu navigation and configuration must use the 'one button interface' (Section 3.2) to obtain input from the user.
- allow the user to create a feed schedule that can allow any number of feed times per day (subject to device memory) to be set to an accuracy of one minute. Each feed should allow the number of rotations of the feed cylinder to be specified.
- save the feed schedule to a formatted plain text file that is reloaded when the program is run, if the file is present. This simulates storing the feed schedule in the device EEPROM memory.
- blank the display after approximately 1 minute if the button is not pressed (to protect the OLED display from burn-in). The user reactivates the display by pressing the button.

3.1 Your development environment

You will not have access to the prototype hardware, however a *simulator* has been created that emulates the function prototypes used by the hardware libraries for various components (screen, clock, button, motor). The C header file `fish.h` contains the set of functions used to access the simulator. The simulator provides a graphical user interface shown in Figure 1.

Debugging information associated with the messages that are sent to the GUI can be shown in the 'Debugging information' part of the window (this area can be deactivated). The simulator also includes additional switchable debugging information that can be displayed on the console (see commented out lines in the `main()` function in the `main.c` file example discussed in Appendix 2).

3.2 One button interface

The hardware has only one button and the plan is to create a 'one button interface' using the button to select functions, navigate menus, and set values. The general idea is to use long and short presses. Long presses select or confirm items and short presses move between items or change values.

[1] driving the motor faster will reduce the torque and may result in the stepper motor missing steps, especially if the feeder is full of food

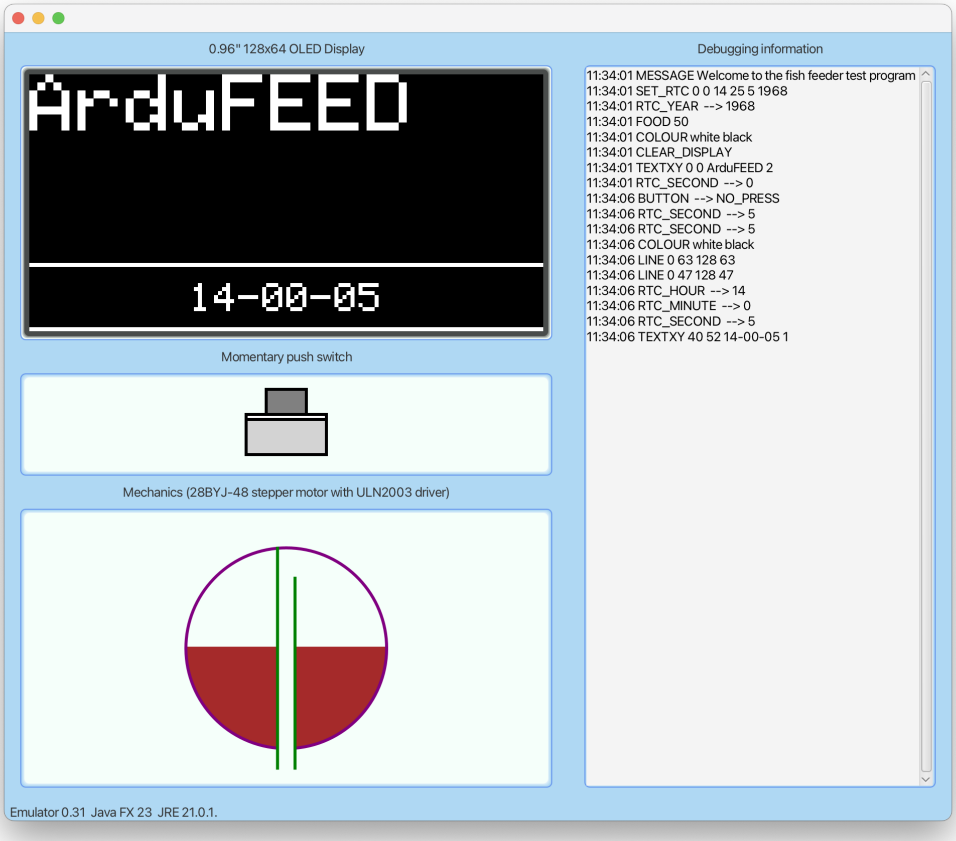

Figure 1: Emulator GUI example

3.3 The emulator functions

The functions have a name prefix corresponding to the associated hardware component (display, clock, motor, button). Calling the functions will result in a message being sent to the GUI and the relevant change being made or information returned. See appendix B for the function signatures and usage information[2].

The GUI has an optional 'debug' window that will show the messages it receives with timestamps so you can monitor what is happening. The stepper motor will also 'miss steps' if you drive it too fast. You can't break the laws of physics so this behaviour is deliberate! Be aware that if you flood the GUI with extremely large numbers of messages they may be ignored or cause the GUI to become unresponsive.

3.4 Feed Schedule file

In the real device the feed schedule will be stored in EEPROM configuration (permanent) memory. For this assignment the simulated hardware does not include the microcontroller, and a text file must be used to store the information. The file should be plain text in a format that is human readable. The exact format is for you to design.

[2]These C functions have similar parameters to the library functions provided by the hardware manufacturer of the devices, and hence it would be a small amount of work to compile the code for the real hardware instead of the GUI emulator.

The GUI simulation includes a function called `clockWarmStart()` that can be used to simulate the real time clock. `clockWarmStart(0)` returns a long integer that can be stored in the schedule file. If that number is passed back to `clockWarmStart(0)` as a parameter the clock will maintain its time keeping capability, i.e. it will keep time as though the program had never stopped[3]

3.5 Your project setup

The simulator and its GUI is a set of C functions (`fish.h` & `fish.c`) connected to a JavaFX application packaged as a custom java runtime (CJR) so no java installation is necessary on the system. Everything is contained in a single folder that needs to be placed in your CLion project.

A CMake project template is available on Blackboard that includes the `fish.h/c` files and a `main.c` simple test program. It is necessary to configure the supplied `CMakeLists` file for your platform by uncommenting the two `include_directories` & `target_link_libraries` lines relevant to your operating system and copy the correct CJR for your platform into the project.

Refer to Appendix 2 for details of the project setup required.

4 Analysis and discussion

This final part of the assignment requires you to discuss how you would change the design and implementation of your assignment if you were asked to re-implement it in C++. You are **not** required to actually implement a C++ version, but you are required to discuss the design you would use and the features of C++ that you would use. For example what aspects of your C design or implementation would be different using C++. The aim is to make full use of object oriented design and other features of C++ that would result in a clean, maintainable, extensible design. You are advised to use diagrams to present your design with explanations as necessary to explain it. You do not need to use formal UML diagrams to describe your design (you may if you wish) however informal diagrams of the design are often a good way to show system architecture and design.

5 Extended features

For those that have completed the requirements, have remaining time, and desire an additional challenge a few marks are available for one or more 'stretch' features.

One example might be to allow a weekly schedule to be configured where different times can be set for any day of the week. Selection of all weekdays and weekend days as a group might also be useful. A related feature would be to select weekend days or week days (Monday - Friday), as a group of times. Other features such as long press the button when the display is off to go directly to the mode selection menu from (to access feed now) or interesting splash screens etc., are all possible.

[3] Notice this number will not change unless the clock is set to a different time

6 The material you must submit

Your solution to this assignment must be submitted on AberLearn Blackboard **before 1pm on Friday 13th December 2023**. The file will take some time to upload so please leave plenty of time. You can resubmit any number of times before the deadline so submit early and update again if necessary. There will be appropriate links in the Assignments part of the course materials for module CS23820. NOTE: This is an individual assignment and must be completed as a one person effort by the student submitting the work.

You must submit a `zip` file containing the following files:

- **The source code of your programs**. You should submit these as your IDE (e.g. CLion) project folder. Make sure that all the necessary files are included. I suggest copying the project into a submission folder and delete the customjre and build folder to reduce the file size before creating the .zip file that includes the code plus all of the submission elements below.
- **A README file** that describes what is what. This is a plain text file (not a MS Word document, not a pdf document). In particular this file must give the names of the various files you hand in and what they correspond to. This must also describe how to build your code, including a mention of standard libraries/packages and the versions you used.
- **A screencast**, showing:
  1. a clean building (in Clion use Build $\Rightarrow$ Clean and then Build) of the code of your programs, with all warnings (hopefully also none — remember in almost every case warnings are really caused by bugs!);
  2. the running of your code, clearly showing *at least* the output from 3 of the supplied NFSF specifications. If you have attempted the extended feature include examples and explain your extension.

  The screencast must include voice to explain what is going on. Also, make sure you choose the correct compilation options so that warnings are given when you rebuild your project for the screencast. Alternatively, this can be one screencast per program. The screencast or casts must run for no more than 10 minutes in total.
- **A document** (maximum 2000 words) that has two sections. The first section called “C program overview” should summarise what you have done for your C program development. Include brief discussion of any notable issues or design features and mention how robust and well tested you believe your program is. The second section called “My C++ approach to the program” should discuss what would be different if you were to use C++ to implement the program as discussed in section 4. Diagrams are likely to be useful in the report.

7 Assessment criteria

The most appropriate “assessment criteria” are those in Appendix AA of the student handbook, namely those for “Assessment Criteria for Development”.

The usual requirements for coding projects apply, namely that programs should be well commented with comments that add real value and do not just, in essence, duplicate code. Programs should have good layout and must use meaningful names for variables, functions

and other identifiers. Finally, part of the evaluation is done on the compilation and whether your code compiles without warnings.

More specifically, the detailed marking scheme for this particular assignment is given in Table 1.

Overall, this assignment is worth 60% of the total marks available for this module.

Table 1: Assessed aspects of your work and their value

What	Value
Code layout readability	5%
Identifier names	5%
Comments	5%
Program quality and design	20%
Program compilation	5%
Program success	25%
C program documentation	10%
C++ design discussion	15%
Extended features	10%
Total	100%

7.1 Implementation requirements

You are expected to make good use of the facilities of the ANSI C programming language.

Your program **must** make good use of **functions** to modularise your code in a sensible way. It should not contain lots of repeated code or very big functions, etc. This will mean thinking carefully about the types of operations and structures you need, and creating reusable functionality where possible.

You should structure your code into **multiple files** with appropriate header files as necessary to modularise your code.

We expect you to sensibly comment your programs, making sure that all comments add real value and do not just, in essence, duplicate code. The programs must have good layout and must use meaningful names for functions, variables, structures, enumerated types and all other identifiers. Single letter identifiers may be suitable where domain convention dictates (e.g. x and y for coordinates) or perhaps for simple small-loop variables or but should be avoided in most other cases.

You may use AI tools such as Copilot if you wish however you must understand all submitted code and be able to explain it in detail in any Authenticity Interview. Please leave any 'prompt engineering' comments you have used that produced larger (i.e. function sized) chunks of code. Also include a discussion in your report explaining which tools you used (if any), how you used them and how useful they were. **Please note that regardless of the tools you used, your code is expected to be well-structured, consistent, well-designed and properly documented.** AI tools often produce code that does not satisfy these criteria without substantial modification. It is better to submit code that partially fulfils the specification and works well rather than broken, badly designed, fragile code (that could therefore be AI generated). Implement your project in small stages and create tests for *each stage* ensuring you keep backups at each milestone to ensure you always have a working version available if necessary.

A Project setup

You can download a template project called `2024-2025_fish_C.zip` that contains a CMake project called `2024-2025_fish_C` containing the files:

- `fish.h and fish.c` – the C interface to the GUI
- `main.c` – example code to use the GUI. Once you have tested the GUI, you will replace this with your own code.
- `CMakeLists.txt` – a minimal build file. You need to uncomment the lines that relate to your OS and architecture. Note that for MS Windows it is also necessary to set a Path environment variable in CLion (as specified in the file).
- An *empty* folder called FishFeederGUI. – This folder will need to contain the fish GUI custom java runtime (CJR) for your platform that you download from blackboard (see below).
- README.txt – this file explains the project setup.

You will also need to allow CLion to "create project from CMakeLists" when it asks because the project download does not contain the platform specific hidden project files.

Before the project will compile and run you need to download the Fish GUI interface package and make a couple of changes to the `CMakeLists` file to configure it for your platform. Download the Fish GUI `customjre` folder for your platform from Blackboard[4]. To be clear you only need ONE of these and it must match the architecture and OS for your machine! The files are:

- customjre_Linuxx64.zip – Linux for 64 bit Intel based machines
- customjre_MacOSaarch64.zip – MacOS on M1/M2 Arm machines
- customjre_MacOSx64.zip – MacOS on Intel machines
- customjre_Windowsx64.zip – Windows on 64 bit Intel based machines

The unzipped file in all cases will produce a folder called `customjre` which must be placed directly inside the `FishFeederGUI` folder of your project. The zip file contains one folder called customjre, but this may end up in another folder dependent on the zip tool. Ensure it is only the `customjre` folder that is placed *inside* the `FishFeederGUI` folder of your project.

Once you have downloaded the relevant material the folder hierarchy should look like Figure 2, where the top level folder is the CLion project folder. Note that CLion will add additional folders (and hidden files) when it opens and builds the project.

Although not necessary for the assignment, the Java code and CJR build scripts, are also available as supplementary information on Blackboard so that you can see what is involved if you are interested in understanding more about Java as well as C!

[4] if your platform is not listed here (i.e. Linux aarch64 we will need to compile a jre on your machine. Please ask nns)

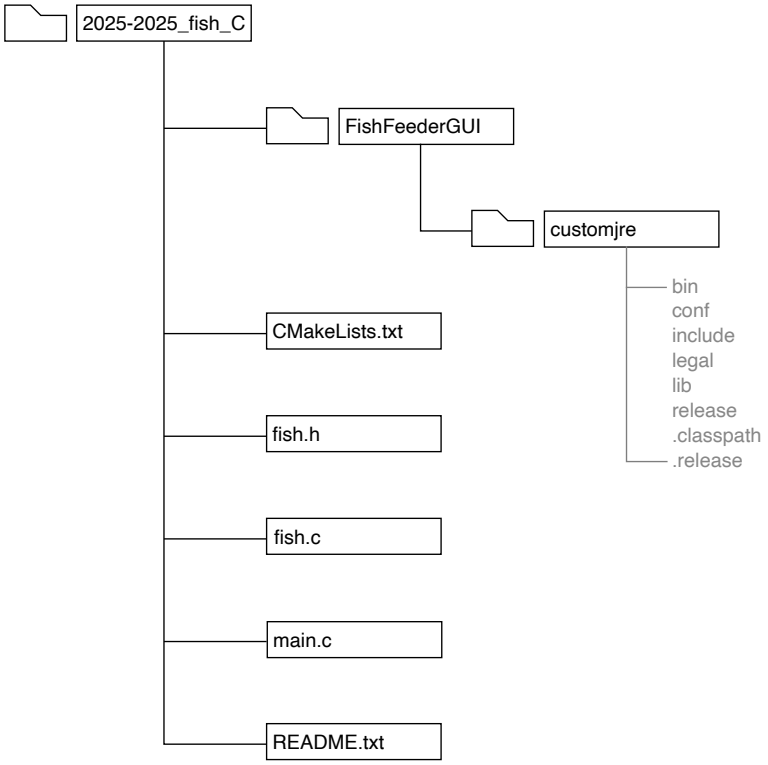

Figure 2: Project initial setup

B Fish feeder emulator functions

This is the `fish.h` header file.

```
/*
 * Created by Neal Snooke on 12/07/2024.
 * This file provides functions for the fish feeder hardware simulator
 * The signatures match (a subset of) those used in the arduino library to access:
 *   DS3201 real time clock chip (virtuabotixRTC.h)
 *   128x64 OLED displays (GOFi2cOLED.h)
 *   plus functions to check the button GPIO input and send pulse sequences to the
 *   ULN2003 stepper motor driver
 *   Additionally a function is provided to simulate the user filling the feeder.
 *
 * The utility functions section includes functions to set up the simulator
 * and control debug output (both in the GUI simulator debug area and on the console)
 */

// display functions for the 128x64 OLED display
void displayClear();
void displayColour(char* fg, char* bg); // set foreground and background colour
                                        // valid options "BLACK", "WHITE", ""(none)
void displayText(int x, int y, char* text, int size); // pixel coordinates; size 1 or 2 only
void displayPixel(int x, int y); // set an individual pixel;
void displayLine(int x, int y, int w, int h); // draw a line between any two coordinates
void displayClearArea(int x, int y, int w, int h); // clear partial display area (to background)

// real time clock (RTC) functions
// set the clock.
void clockSet(int sec, int min, int hour, int day, int month, int year);
// retrieve a specific element of the time/date
int clockSecond();
```

```
int clockMinute();
int clockHour();
int clockDay();
int clockMonth();
int clockYear();
int clockDayOfWeek(); // Sunday = 0, Monday = 1, etc
// to maintain a clock between executions of the emulator save the value returned by this function
// when given 0. restore the clock by calling the function with the previously saved value.
// the clock will have continued to keep time.
// note the result value will not change unless the clock is set to a different time with clockSet()
long clockWarmStart(long offset);

// mechanical feeder functions
void motorStep(); // rotates the feeder container one step (1 degree)
void foodFill(int foodLevel); // set food level. range 1 - 60%

// button function
// returns one of "SHORT_PRESS" "LONG_PRESS" "NO_PRESS"
char* buttonState(); // note caller must dispose of char* result

//------------------
// utility functions
//------------------

// info area function to display information in the GUI debug info area (if enabled)
void infoMessage(char *text); // display an information message

// this function is where the main users code goes and must be implemented to use the simulator
void userProcessing();

// jni functions for the JavaFX fish feeder simulation GUI
// must be called once only and in sequence since jni_setup spawns a
// new thread for user processing allowing the GUI to run in the main thread.
int jni_setup(); // setup the JavaFX GUI and then run userProcessing() once GUI is initialised
int java_fx(); // initialise the JavaFX GUI - must be called after jni_setup. Returns when GUI quits

// delay for a specified number of milliseconds
int msleep(long msec);

// it is possible to output various levels of debug info from the Fish GUI Emulator Java and C code
// the following constants are used to select what to output to the console log.
// the debug level is a single integer,
// with information selected by bitwise OR of the following constants
extern const int THREAD_NAME; // append the thread name to all log messages
extern const int THREAD_ID; // append the thread id to all log messages
extern const int METHOD_ENTRY; // method entry and exit messages
extern const int JNI_MESSAGES; // Java Native Interface messages
extern const int JFX_MESSAGES; // JavaFX processing messages
extern const int GENERAL;
extern const int STACK_INFO; // stack trace information
extern const int GUI_INFO_DEBUG;

extern int  log_level; // global that stores the current log level setting

// add a log entry for a given log level. l is one of the constants specified above
void logAdd(int l, char* message);

// select a log level to output. parameter is a bitwise OR of the above constants
void logAddInfo(int i);
// stop logging a specified level. parameter is one of the constants specified above
void logRemoveInfo(int i);
```