Callum Stanford-Hoare(cas113)

Fish Feeder C Assignment Documentation

This is the report for the fish CS23820 Fish Feeder Firmware assignment. It should be found in the same folder as the readme and screencasts as well as the actual code in the present folder.

Part 1: C Program Overview

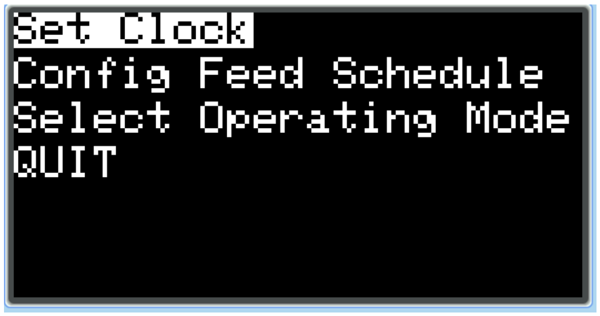
For my C program development, I decided to decompose the problem into these areas: menu navigation, menu functionality, scheduling, file handling, setting the clock and finally tidying up functionality. I tackled the problem in the listed order first by developing a very basic starting menu that could be traversed which I copied into a sub menu soon giving the realisation that my solution had to be more generic. Thus a generic menu was implemented in which each menu passes what it wants to display to. The next part was the functionality of these menus where I opted to implement a struct for each menu where you can list the text to be displayed and its associated function pointer. Selecting this in the program will then trigger said function. This was rather smooth sailing for most basic functionality like getting into the configuration menu, and included the implementation of a generic menu but for selecting values. This could be used for instances like setting the clock or creating schedules.

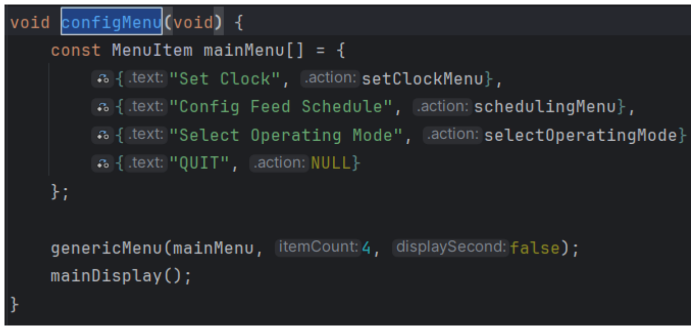
Scheduling was perhaps the second most lengthy thing to implement as it took much planning and trailing to be able to create a dynamic array inside a struct that held structs, much was learnt about reallocation of memory through this. Thankfully file handling was rather rudimentary and a nice ‘break’ before clock setting. I describe it as a break as clock setting was where a lot of my problems started as this section also included the logic to actually feed the fish.

I was getting where after starting the program I could not printf anything to console nor were any of my function pointers working in the genericmenu. I eventually resolved this by exporting the logic to a new function and then correcting some logic errors. Setting the clock also made me realise that I needed to make use of the warmclockstart function where only upon reading the brief did I discover the value needed to be saved to the file. Another obstacle was being able to get the next feed time correctly as the program needed to be able to change this time based on if the time had already passed because the program had been closed.

After the program had been "done" I decided to add some quality of life features here and there. For instance on the generic menu for setting numbers I added a wrap around feature to make it easier to go through the numbers. I additionally tidied up the menus, like having the arrows in the middle of the GUI and a bit of text to inform the user that the button logic they had been so used to using had now been reversed. Additionally I went along with some code suggestions CLion was giving me such as removing unused includes and changing comparison of string literals to use string comparisons rather than what I had incorrectly implemented.

A good exercise for describing the flow of my program would be to describe the code directly.





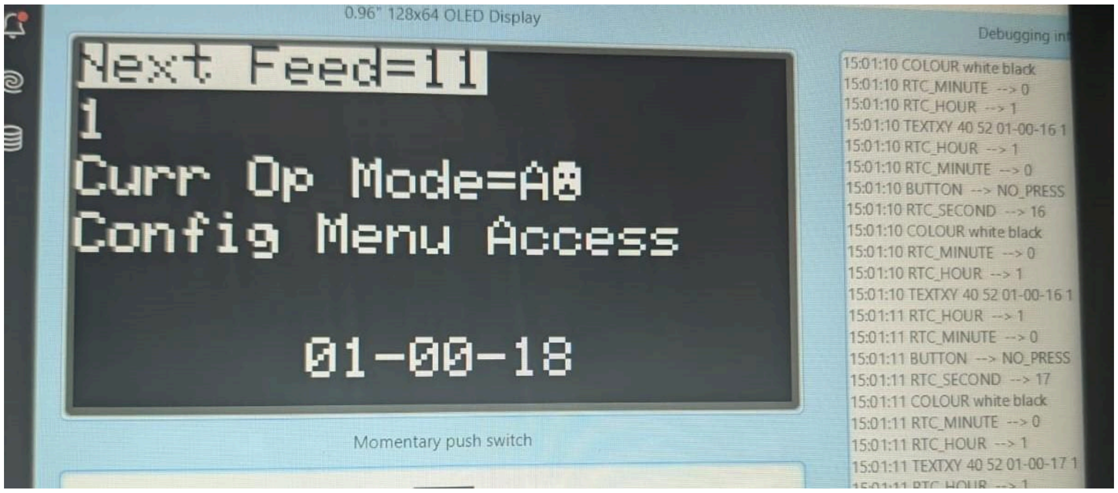
```
void configMenu(void) {
    const MenuItem mainMenu[] = {
        {.text: "Set Clock", .action: setClockMenu},
        {.text: "Config Feed Schedule", .action: schedulingMenu},
        {.text: "Select Operating Mode", .action: selectOperatingMode}
        {.text: "QUIT", .action: NULL}
    };

    genericMenu(mainMenu, itemCount: 4, displaySecond: false);
    mainDisplay();
}
```

The above screen captures give the picture of the configuration menu and gives outline to how the program works its menus. Each menu is its own function and holds a struct. This has the text we want to display and a function pointer. In practice if the user long presses it will trigger the function. Where it reads NULL in the place of a function pointer, the program interprets this as wanting to quit the generic menu function. It can then go back to the function before it. Essentially making each menu a chain where each menu is reliant on its before and after. The only exception being the “splash screen” which is used to initialize the program’s values and then actually start the program once the user inputs something.

Around the program we want to store and retrieve values. Values such as the number of feeds since auto or the operating state. Due to the nature of the menus I opted for using a state file which holds a number of statically defined values and associated getters and setters. This way we can access or manipulate values anywhere we need which is essentially everywhere. The alternative would have been to pass values down the chain but this would have made for functions with excessively large parameter amounts.

One error that was encountered doing this main menu functionality can be seen in the screencap below:



```
Next Feed=11
1
Curr Op Mode=A☹
Config Menu Access

     01-00-18
```

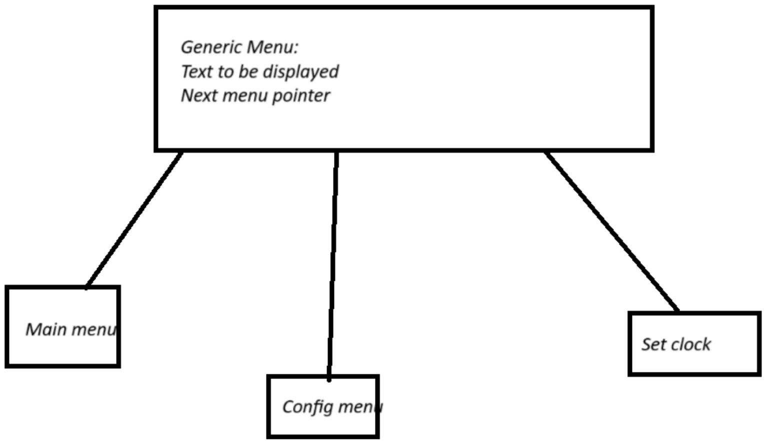
```
15:01:10 COLOUR white black
15:01:10 RTC_MINUTE --> 0
15:01:10 RTC_HOUR --> 1
15:01:10 TEXTXY 40 52 01-00-16 1
15:01:10 RTC_HOUR --> 1
15:01:10 RTC_MINUTE --> 0
15:01:10 BUTTON --> NO_PRESS
15:01:10 RTC_SECOND --> 16
15:01:10 COLOUR white black
15:01:10 RTC_MINUTE --> 0
15:01:10 RTC_HOUR --> 1
15:01:10 TEXTXY 40 52 01-00-16 1
15:01:11 RTC_HOUR --> 1
15:01:11 RTC_MINUTE --> 0
15:01:11 BUTTON --> NO_PRESS
15:01:11 RTC_SECOND --> 17
15:01:11 COLOUR white black
15:01:11 RTC_MINUTE --> 0
15:01:11 RTC_HOUR --> 1
15:01:11 TEXTXY 40 52 01-00-17 1
15:01:11 RTC_HOUR --> 1
```

I had encountered an error with what I am guessing has to be overflows when attempting to do 2 strcat and sprintf conversions giving me the above headache. This was eventually resolved by enlarging the char arrays to larger dimensions so as to not get any weird character errors, like whatever that face is and wherever it came from.

On the topic of robustness of my program I have not been able to test everything. I am certain there is no case present that completely breaks the program through the users doing. I am aware of some UI peculiarities however for starters, one of which is when accessing any generic menu for numbers if you press the left arrow first signally to decrement the value the value will be highlighted. Another thing is after a full feed, if the user is currently on the main display they will have to enter the configuration menu and then go back to the main display for the values to be updated. The fix for this would be to just simply recall the mainDisplay method but this poses the problem that if the user is actively using the program and then a feed occurs, all of their progress will be for nothing. I have currently implemented a sort of band aid solution by having a refresh option for the main page. And a final area that is logically inconvenient is that adding a new schedule that should be the next schedule does not change the next schedule time to be that time, instead the next schedule time is simply locked as whatever the program decided was the next time on load. The current solution for this is to just restart the program and have the initialization method run up again which is not great at all, however due to complexity of the solution I can't currently decipher on where I would check that a new addition should be the next time. I'm sure a number of bugs still persist that have yet to be found, such as the inability to set the time to be 12:XX. Bugs like this would require actual user testing to find out issues deep within the code.

Part 2: My C++ approach to the program

The move to C++ would grant a number of new techniques that inherently come with object oriented programming. The first move would be instead of having each menu independent in its structure we could derive structures from a menu class where we explicitly define the variables. Another idea building on this would be to have the genericmenu inherited here perhaps allowing us to have more specialized menu functionality where it is needed.



This would be the basic idea of deriving from a single class. That way if we wanted to add something to the a menu like each menu having a pointer to the previous menu as well this could be easily implemented, else in the current C program solution making any universal change to the menu system would be a laborious task of copying and editing a good portion of the functions in the code. Also to note alongside this implementation we would have needed to use constructors and deconstructors in order to create instances of these menus and then get rid of them after they have been sorted through. This is why it may be good to implement pointers to previous functions as that way they can be deconstructed due to not being used anymore.

C++ also authorises use of a wider library of functionality, perhaps implementation of some data structures such lists in place of manually allocated arrays would make implementation of our list of schedules far more easier. Alternatively smart pointers in place of manual memory management would also be a lot more effective and would give less of a harder task of implementation than what is currently present, that being the manually allocated malloc statement for initializing the list of structs array as well as the reallocation needed every time an item is added, it would also be presumably more memory efficient to have the compiler sort it rather than the developer.

Range based loops could be used to great effect in cooperation with our new data types as for example with our schedule list we can just iterate over each object to decide which should be the next element rather than having to maintain different variables. Moreover since C++ enables strings as a data type, which would be of benefit to implementation as it gives access to a wider library of methods. One area that we could use string and string manipulation would be in the main display function in order to get the various elements that we wish to display. At present the code is quite long relying on a series of sprintf and string concentrations, it would be a world's easier time to simply use strings and concatenate things like integers onto the needed text. It may also avoid the need for having long arrays to avoid the weird error that was seen in the discussion of my current code implementation.

One element that I did explicitly wish to implement in my code but couldn’t was a bit of function overloading. Mainly in regards to the rotation function where I have had to copy the code and change the function name in order to be able to have a rotation for an amount of cycles and a single rotation due to the requirements of the program. With C++ I could have it as the same name but with different parameter signatures, this would then enable less code duplication to be present within the program and is a concept that i'm sure can be applied elsewhere, if other code duplication can be found.

Exception handling would be a great addition to our file handling section as this would better highlight attempts to write or read to and from a file. This would involve using a try catch method to check if the file is there rather than relying on the C version of relying on an if statement with simple text outputs to say that the file is not there.