22COA202 Coursework

F210502

Semester 2

1 FSMs

My first state is waiting for the user to type 'X'. I have included this in my set-up loop because it only needs to be checked once when the code is first run. After this I put the main code in loop.

The first state is "WAITING_FOR_INPUT". This is essentially the device in idle mode. It'll display devices (if there are any) and wait for input from the serial monitor. If a user enters a command, it will be processed here. I could've potentially added another state called "PROCESSING_COMMAND" here, for example. However, processing the command in the same state worked, so I didn't change it.

From this position the device has three states it can transition to. These are HOLD_SELECT, SCROLL_UP and SCROLL_DOWN. These all depend on the buttons the user presses.

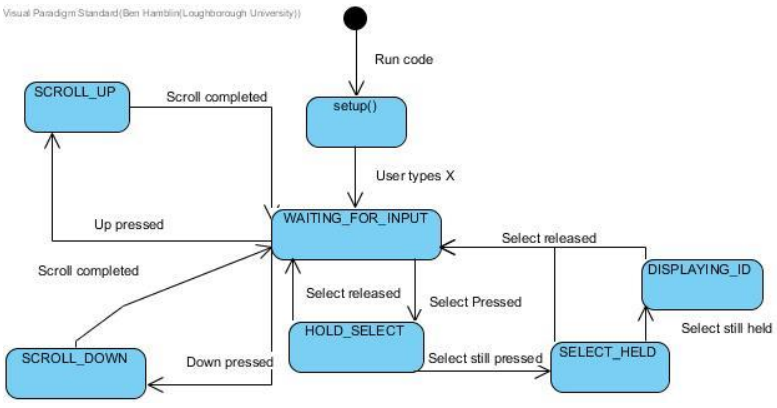
HOLD_SELECT occurs when the user presses the select button. We also enter a sub-FSM here. It stores the time the button was initially pressed. We transition to a state "SELECT_HELD" if the button is still pressed.

SELECT_HELD is used as a crossroad. We transition to the idle (WAITING_FOR_INPUT) state if the select button is released. Otherwise, we display the ID and free memory if it has been over a second and then transition to our final substate.

The final substate, DISPLAYING_ID, checks if the select button has released. If it has then we transition back to WAITING_FOR_INPUT.

SCROLL_UP occurs when the user presses the up button. If it is possible, we then "scroll up" by displaying the previous device. Then we transition back to WAITING_FOR_INPUT.

SCROLL_DOWN occurs when the user presses the up button. If it is possible, we then "scroll down" by displaying the next device. Then we transition back to WAITING_FOR_INPUT.



2 Data structures

Macros:

- Colours – assigned colour names to hex codes so I can easily set the backlight colour of the LCD screen.
- States – assign states to numbers to save some run time SRAM space

Enums:

- DeviceTypes – a set standard of types (Speaker, socket, light, thermostat, and camera) to help with user validation and printing to the LCD. When the user enters a command, I can check if the device type is valid. If it is, I can assign it to a macro. Then when I print the device to screen I will use this macro.

Classes:

- Device class – new instance for each device added to the global store. This class is used to create and manage devices with a device ID, type, location, state, and power. It has getter and setter methods for each attribute.
- Device manager class – deviceManager is the instance. Has two properties: a pointer to an array of devices and an integer which has the number of devices. Includes addDevice() and removeDevice() methods to update global device store.

Constants:

- Max Devices – constant which limits the number of devices that can be added. This is useful as if I want to change this number, I only must edit one line.
- Bytes – for the UDCHARS extension, my characters are stored in byte arrays that are not changed during run time

3 Debugging

During development I would often print lines to the serial to check if a condition has been met / a loop has been entered. However, I would delete these as I went on.

When testing using the automated python program, I found inputting the commands with no delay caused an issue with my display. I had to add a delay into the python code so that it had time to process each command. 2 seconds was enough, but in hindsight I would have liked for my Arduino to be more efficient than this.

One issue I faced was: when adding a new device with the same ID, do I change the type and/or power values? In the end I opted for keeping the power value if the same device type is added. However, if a new device is added with a different type, I removed the power value, as you have essentially never given that specific device a power.

4 Reflection

I would like my code to be more efficient. My Arduino can be slow to load instructions and require a few seconds to process each command.

This may be because I use a lot of strings to make use of the built in functions and their mutability. However, I could use char arrays instead to increase memory efficiency and performance.

Certain functions also may limit performance. For example, I use a selection sort to order my devices by alphabetical order. I could have implemented a more efficient algorithm such as quick sort. C has a qsort() function which I could've looked into further.

In general, my code is also quite chaotic. During development my FSM designs changed quite a bit, so my states aren't all as useful as each other. I think I could simplify the current ones and potentially reduce the amount I have. For example, I only really need 2/3 of "HOLD_SELECT, SELECT_HELD, AND DISPLAYING_ID".

Also, I could have used a struct instead of a class for my devices, but I felt more comfortable and familiar with classes.

5 UDCHARS

Using this website, I created my custom up and down arrows, as well as a degrees symbol for temperature. The website allows you to "fill in" squares in an 8x5 grid which acts as drawing a character. From here I copied the binary representation of this and stored it as a constant byte. E.g.

```
byte downArrow[8] = {

  B00100,
  B00100,
  B00100,
  B00100,
  B00100,
  B10101,
  B01110,
  B00100
};
```

Once I declared the characters in my code, I used the lcd.createChar() method so that they can be displayed on my LCD screen.

I also used this same method for my degrees symbol. The code on the right shows me accessing it and concatenating it with my temperature value.

```
char degrees = byte(3);
result = result + degrees + "C";
myDevice->setPower(result);
```

6 FREERAM

Using the MemoryFree library from Arduino Playground, I had two options to calculate the FreeRAM. One, called freeRam(), was simpler and subtracted the heap pointer/beginning of the heap from the address of a new local variable v. The second, freeMemory(), was more complicated and it takes into account the linked list of free memory blocks. It calculated the free memory by subtracting the stack pointer from the heap pointer/beginning of the heap. I believe the second function is more accurate and so have used it. Although it is slightly more complicated, I believe the difference in performance will be negligible. As a result, I have used the freeMemory() function to calculate the free memory. I simply then call the function and lcd.print the returned value at the same point the ID is displayed on screen. I have left the alternate function commented out.