24COA202 Coursework

*F418623*

Semester 1

# FSMs

A diagram of a software system

Description automatically generated with medium confidence

Shown above is a diagram of the Finite State Machine (FSM) that is used in my implementation. Due to the unfortunate simplicity of the model, I created a less finite diagram below to show the logic and operations behind many of the states to help give you a better understanding of the system’s processes.

**A diagram of a diagram

Description automatically generated**

# 2 Data structures

### Structures (structs):

Each employee is stored in a structure, grouping all the required variables together for each employee. These variables include ‘ID’, ‘jobTitle’, ‘pension’, ‘grade’ and ‘salary’, representing each employee’s ID number, job title, pension status, paygrade and salary respectively.

### Arrays:

The structure of each employee’s data is stored in an array of a constant length named ‘employees’. This means that the data of each employee is stored in one place such that it can easily be searched through, added to or removed from, all of which are key features of the payroll system, making the array a perfect fit for the implementation.

### Constants:

A constant variable named ‘EMPLOYEECOUNT’ is utilised to store the maximum number of employees that can be stored on the system. This value is stored as a constant to allow for a quick and easy change in maximum number of employees, without having to change multiple lines of code where the constant is in use. This was especially useful once the program was completed, allowing me to increase the count to a more realistic number of employees once the system’s functionality was completed.

**2 Functionality that alters global structures**

### Adding employees to the system (ADD Command):

Updates a currently vacant structure on the ‘employees’ array by filling it with relevant data about a new employee added freshly to the system.

### Deleting employees from the system (DEL Command):

Wipes all current data about an employee by restoring each variable in the struct to a default value such that the index in the array can be repopulated later by a new employee added.

### Altering employee information (CJT, GRD, SAL, PST Commands):

Changes a certain variable within an employee’s structure depending on the command specified.

### Using UP and DOWN buttons on the Arduino:

Increments or decrements the ‘currentScreen’ variable so that when the program completes its loop it will display the details of the employee above or below accordingly.

# 3 Debugging

The biggest bug I encountered in the system was one to do with the idle main phase. After exiting the sync phase, the Arduino screen would turn red and constantly print hundreds of my custom arrow characters and many other unknown characters along with messages that were supposed to be printed to the serial monitor. I found that this behaviour went away when an account or more was added, and the system was running. After a while I figured that during the idle state of the main phase, many of the if statements could be entered unintentionally. To get around this I had to use a Boolean variable called ‘begun’ to block any unintended access of important sections of code when the program was in the idle state.

# 4 Reflection

Overall, I am quite happy with how my program has turned out. I think the development of the sync phase went very smoothly however it did take a few days pondering and planning how best to implement the main phase of the system. After this slow period however, I certainly got the ball rolling, and I only ran into a few problems within the main phase. Initially I struggled to get the buttons to work however after printing out what the ‘lcd.readButtons()’ function recorded with each click I was able to move past it promptly. However, after a few minor bugs and a major bug as written in the section above, I was able to finish the basic implementation. If I were to develop an employee system again using the Arduino I would certainly be less dependent on Strings. I did initially try using a character array in the sync phase however I found it to be a lot more difficult to use and decided that using strings would be a lot more time efficient for my creation of the program. Thus, if I had more time and with the knowledge of C++ that I now have I would utilise character arrays over strings to save me memory space. Furthermore, I would create an acceptance state or a final state for the FSM implementation perhaps prompted by a Serial input that would end the program completely as I feel that it would round of the program much better than the current system.

# Extension Features

# 5 UDCHARS

    lcd.setCursor(0, 0);

    if (currentScreen == 0){

      lcd.print(" ");

    }

    else{

      lcd.write((uint8\_t)1);

    }

    lcd.setCursor(0, 1);

    if ((currentScreen == EMPLOYEECOUNT)||(employees[currentScreen+1].grade == -1)){

      lcd.print(" ");

    }

    else{

      lcd.write((uint8\_t)0);

    }

The code above shows how my custom characters are used in the system. If statements are used to check if there is an employee account above or below the one currently being displayed, printing an up arrow if there is one above and printing a down arrow if there is one below. The code for initialising these characters is shown below.   
  
byte upChar[] = {

  B00100,

  B01110,

  B10101,

  B00100,

  B00100,

  B00100,

  B00100,

  B00100

};

// Down arrow custom character

byte downChar[] = {

  B00100,

  B00100,

  B00100,

  B00100,

  B00100,

  B10101,

  B01110,

  B00100

};

And below shows the above arrays being converted into actual characters within the setup block.

  lcd.createChar(1, upChar);

  lcd.createChar(0, downChar);

# 6 FREERAM

#include <MemoryFree.h>

Installed a MemoryFree.h library from <https://github.com/maniacbug/MemoryFree> and moved its files into my Arduino IDE’s libraries folder, allowing me to include this file.

// Code regarding the SELECT button being held down

  if (buttons == 1){

    if (!selPressed){

      startSelect = millis();

    }

    selPressed = true;

    // Ensuring that select has been held for over a second before f number is displayed

    if (millis() >= startSelect +1000){

      lcd.setBacklight(7);

      lcd.setCursor(0, 0);

      lcd.print("F418623");

      lcd.setCursor(0, 1);

      // Used FreeMemory.h library here to find the number of bytes remainuing in RAM

      lcd.print(freeMemory());

      lcd.print(" Bytes remain");

The code above shows how the file is used within my SELECT button functionality to display the Arduino’s remaining memory.

# 7 SCROLL

Below shows the code and functionality behind the scroll functionality, used when displaying longer job titles. In order to get a smoother look, I decided to scroll the job title by 1 character each half second.

void loop() {

  static int scrollStart = 0;

  static int scrollEnd = 7;

The above snippet shows the initialisation of the scrollStart and scrollEnd variables, but it is important to note that they are set to 0 and 7 again (as shown above), each time a new profile is displayed using the up and down buttons.

    // Printing any shorter job titles.

    if (employees[currentScreen].jobTitle.length() <= 7){

      lcd.print(employees[currentScreen].jobTitle);

    }

    // Scrolling mechanism extension for longer job titles.

    else if (millis() - timeScroll >= 1000){

      scrollStart +=1;

      scrollEnd +=1;

      timeScroll = millis();

      if (scrollEnd > employees[currentScreen].jobTitle.length()){

        scrollEnd = 7;

        scrollStart = 0;

      }

    }

    else{

      lcd.print(employees[currentScreen].jobTitle.substring(scrollStart, scrollEnd));

    }