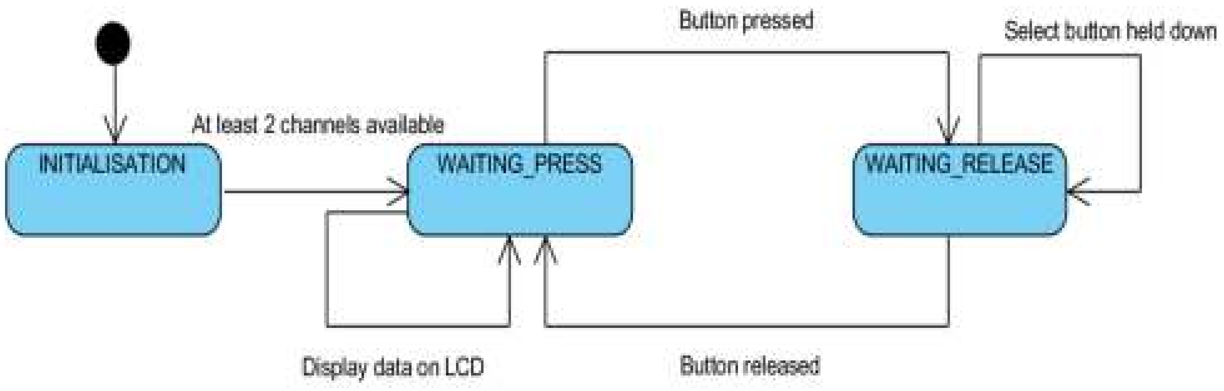
21COA202 Coursework

F121572

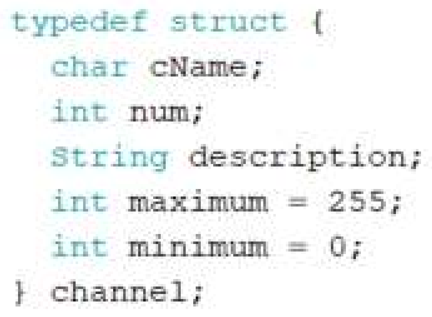
Semester 2

1 FSMs



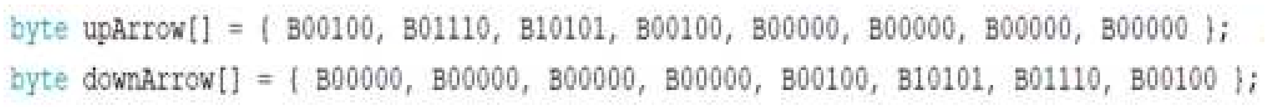
My finite state machine is simple and largely inspired by the lab 4 worksheet. In the INITIALISATION state, input is taken from the serial monitor until there are at least 2 channels. When there are at least 2 channels available, the state is changed to WAITING_PRESS, where the program is able to take input and update the LCD display depending on that input. When a button is pressed, the state is changed to WAITING_RELEASE, where the time of the button press is considered. If the length of the SELECT button press is more than 1 second, it will execute the code within its respective block until the button is released, at which point the state changes back to WAITING_PRESS, and the actions of the other buttons are executed. From there the program will run indefinitely, switching between these 2 states.

2 Data structures



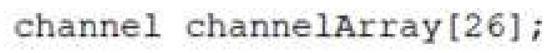
```
typedef struct {
  char cName;
  int num;
  String description;
  int maximum = 255;
  int minimum = 0;
} channel;
```

I created a structure named "channel" which is able to store the relevant details of a channel in 1 simple area. These details are easily accessible which means that it is efficient when reading or writing to them.



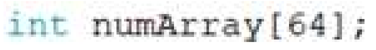
```
byte upArrow[] = { B00100, B01110, B10101, B00100, B00000, B00000, B00000, B00000 };
byte downArrow[] = { B00000, B00000, B00000, B00000, B00100, B10101, B01110, B00100 };
```

I used arrays of bytes in order to store the custom arrow symbols for the UDCHARS extension. I was able to easily create these from the chareditor website.



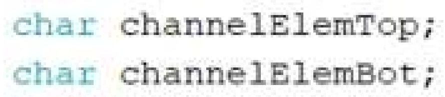
```
channel channelArray[26];
```

In order to store all of the potential channels that could be used, I created an array of type "channel" of length 26, enough space for all the letters of the alphabet.



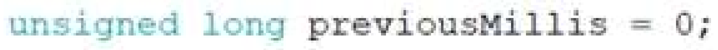
```
int numArray[64];
```

For the RECENT extension I created an array of integers of length 64, which would be able to store the most recent 64 numbers entered. This array would essentially work like a queue, where the oldest item is removed, the list is shifted left and the newest item is added to the front.



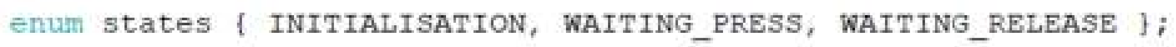
```
char channelElemTop;
char channelElemBot;
```

I created 2 variables of type "char" to store the channel names to be displayed on the top and bottom row of the LCD respectively. I opted for "char" instead of "String" so that memory would be utilised more effectively.



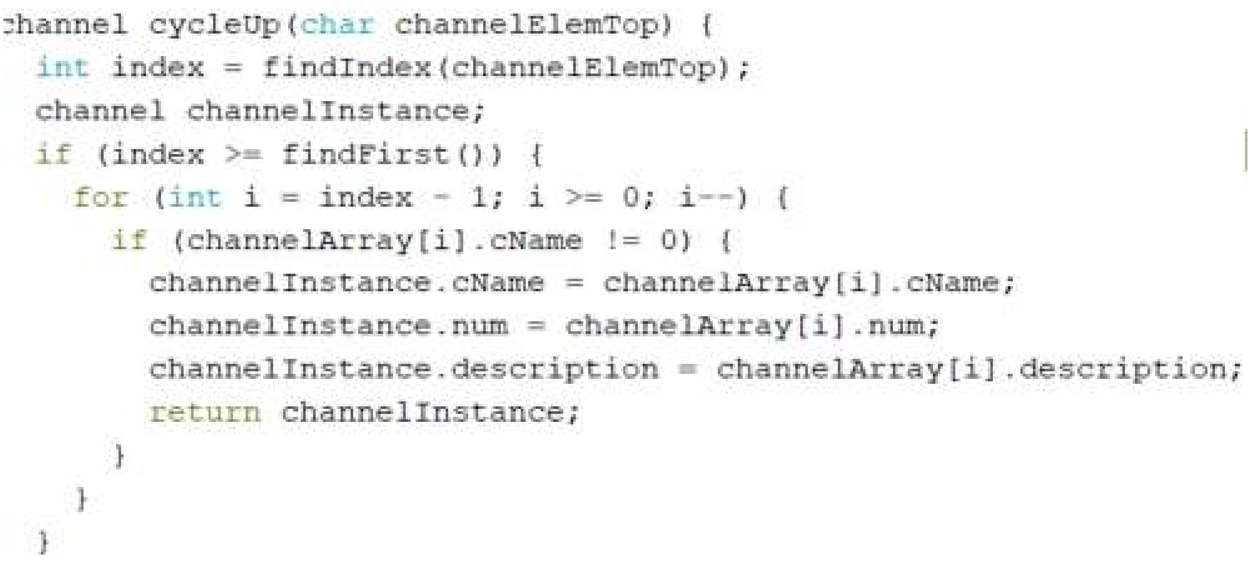
```
unsigned long previousMillis = 0;
```

I used a variable of the "unsigned long" type to store the time for the SCROLL extension. I used this rather than a variable of type "int" because the unit of time would be milliseconds, which means that a 2 byte integer would quickly run out of memory as the program runs.



```
enum states { INITIALISATION, WAITING_PRESS, WAITING_RELEASE };
```

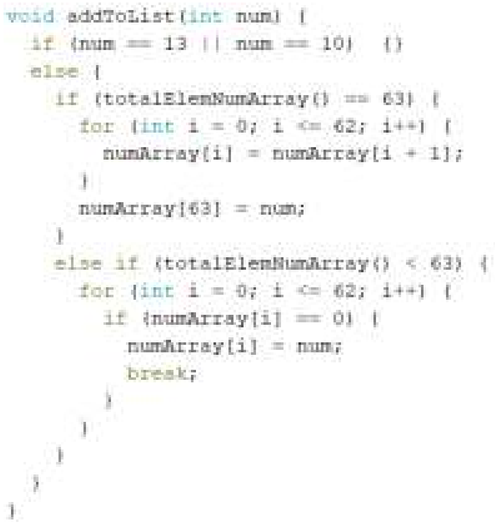
I used the type “enum” to apply numbers to my states, which would be necessary for my finite state machine.



```
channel cycleUp(char channelElemTop) {
  int index = findIndex(channelElemTop);
  channel channelInstance;
  if (index >= findFirst()) {
    for (int i = index - 1; i >= 0; i--) {
      if (channelArray[i].cName != 0) {
        channelInstance.cName = channelArray[i].cName;
        channelInstance.num = channelArray[i].num;
        channelInstance.description = channelArray[i].description;
        return channelInstance;
      }
    }
  }
```

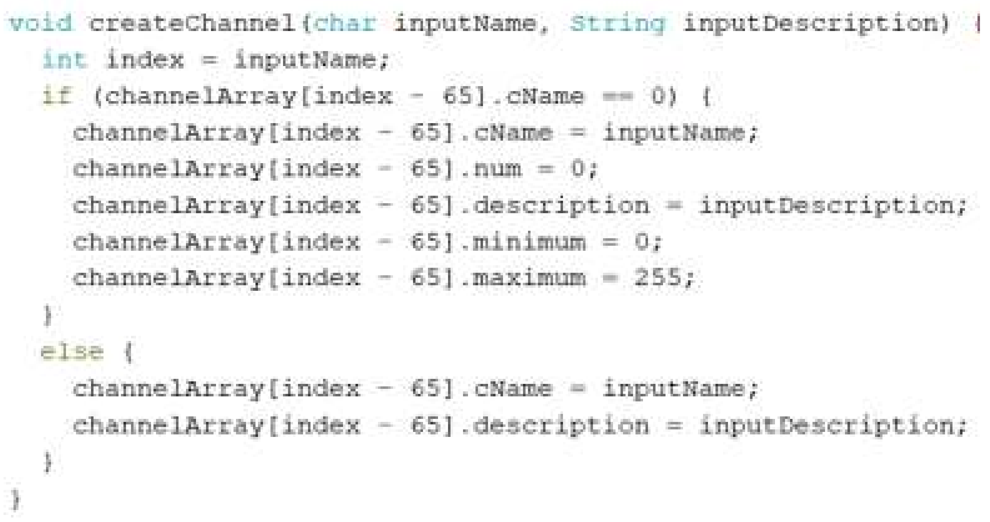
The above function returns an instance of the “channel” structure based on the channel name that was taken as a parameter. Subsequently this channel instance can be accessed in the loop function and data can be taken from it. I decided to implement this because I needed to return data of different types from one function, and a structure allowed me to do that.

Updating data structures



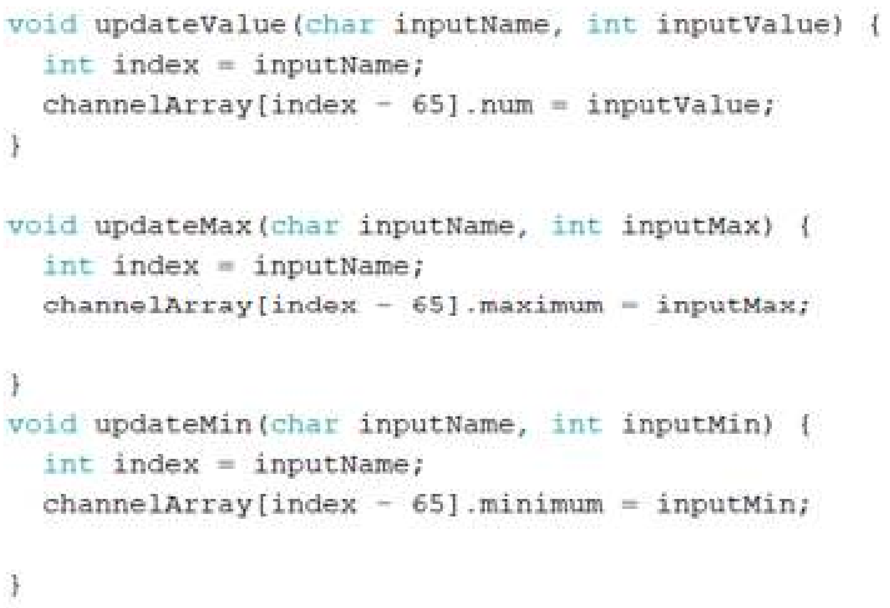
```
void addToList(int num) {
  if (num == 13 || num == 10)  {}
  else {
    if (totalElemNumArray() == 63) {
      for (int i = 0; i <= 62; i++) {
        numArray[i] = numArray[i + 1];
      }
      numArray[63] = num;
    }
    else if (totalElemNumArray() < 63) {
      for (int i = 0; i <= 62; i++) {
        if (numArray[i] == 0) {
          numArray[i] = num;
          break;
        }
      }
    }
  }
}
```

The above function takes the most recent number inputted as a parameter and subsequently checks if the numArray is full. If it is, then it shifts the list left and adds the new number to the end. If not, it goes through the list and finds the first empty index and places the number there.



```
void createChannel(char inputName, String inputDescription) {
  int index = inputName;
  if (channelArray[index - 65].cName == 0) {
    channelArray[index - 65].cName = inputName;
    channelArray[index - 65].num = 0;
    channelArray[index - 65].description = inputDescription;
    channelArray[index - 65].minimum = 0;
    channelArray[index - 65].maximum = 255;
  }
  else {
    channelArray[index - 65].cName = inputName;
    channelArray[index - 65].description = inputDescription;
  }
}
```

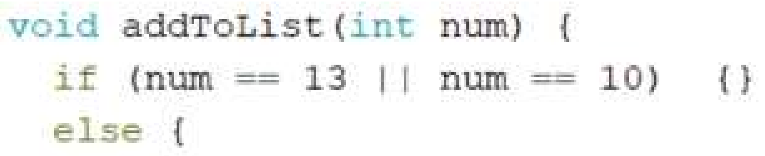
This function takes the channel name as a parameter, and based on the position of the letter in the alphabet, it applies the relevant details to its respective empty structure in the channelArray. If the structure is not empty, then it rewrites the old description to the new one.



```
void updateValue(char inputName, int inputValue) {
  int index = inputName;
  channelArray[index - 65].num = inputValue;
}

void updateMax(char inputName, int inputMax) {
  int index = inputName;
  channelArray[index - 65].maximum = inputMax;

}
void updateMin(char inputName, int inputMin) {
  int index = inputName;
  channelArray[index - 65].minimum = inputMin;

}
```

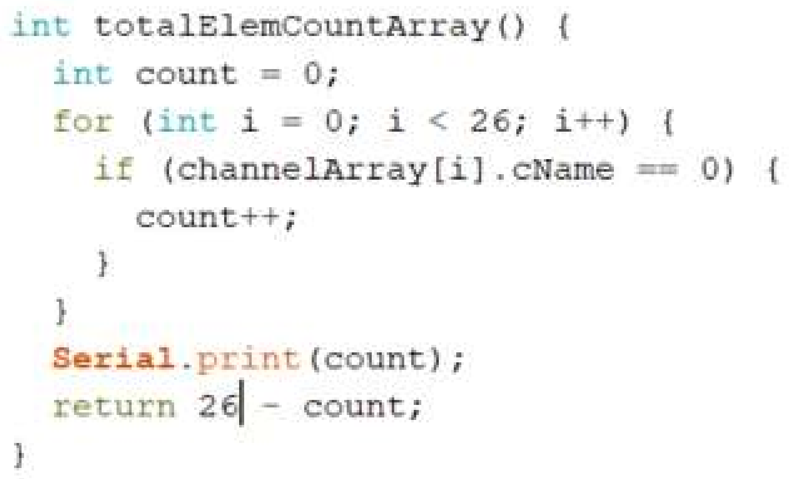
The above series of functions is designed to rewrite the old values to the new ones that are taken as parameters.

3 Debugging



```
void addToList(int num) {
  if (num == 13 || num == 10)   {}
  else {
```

The above code is part of the RECENT extension, and was designed so that if a CR or LF was read, then the program would ignore it. I created this because previously I had issues with 10 being unwittingly added to the numArray. This should not be an issue anymore.



```
int totalElemCountArray() {
  int count = 0;
  for (int i = 0; i < 26; i++) {
    if (channelArray[i].cName == 0) {
      count++;
    }
  }
  Serial.print(count);
  return 26 - count;
}
```

This is an example of code that was causing problems, simply due to the fact that the for-loop was cycling 27 elements of the channelArray instead of 26. When debugging, I often printed the values of variables to the serial monitor to understand the program flow better, which led to me realizing this issue.

4 Reflection

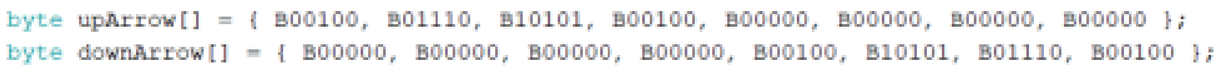
Overall I am quite pleased with my final product. I believe that I have ensured that the main part works well in every aspect, and I have completed most of the extension tasks as well. However, there are several areas in which I could have improved.

I am particularly pleased with the effectiveness of the array of structures as it allows for efficient storage and access of channel data. I was also able to create instances of channels and place data from there into the channel array easily. Another positive was that I was able to devise a system with the channelArray that would place each channel in alphabetical order automatically, without me ever having to apply a complicated sorting algorithm I am also satisfied with my program's memory usage, since I am left with over half of SRAM available, which is enough for my program's extra storage needs.

My main regret was that I was not able to complete the HCI and EEPROM extension, so if I had more time I would have tried to finish those. My second issue was that my SCROLL extension was not implemented as well as I would have liked, since I was forced to make it scroll every 100ms instead of every 1000ms. I believe it was because the program was not going through the timing if-statement correctly, but I was still not able to find a solution. This was the one issue that I spent hours on and did not make any progress with, which was infuriating. In retrospect, if I used my time more wisely perhaps I could have taken some sort of radically different approach instead of wasting time on many failed ideas.

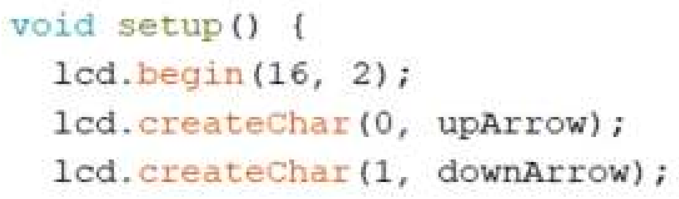
I also thought about implementing a more complicated finite state machine (which perhaps could have solved the SCROLL issue mentioned above) but I felt that the FSM used in the lab 4 sheet was sufficient for my purposes, and my program works well in every other aspect. If I could change it I would maybe add specific states for each button action and a state just for dictating how the data is displayed on the LCD.

5 UDCHARS



```
byte upArrow[] = { B00100, B01110, B10101, B00100, B00000, B00000, B00000, B00000 };
byte downArrow[] = { B00000, B00000, B00000, B00000, B00100, B10101, B01110, B00100 };
```

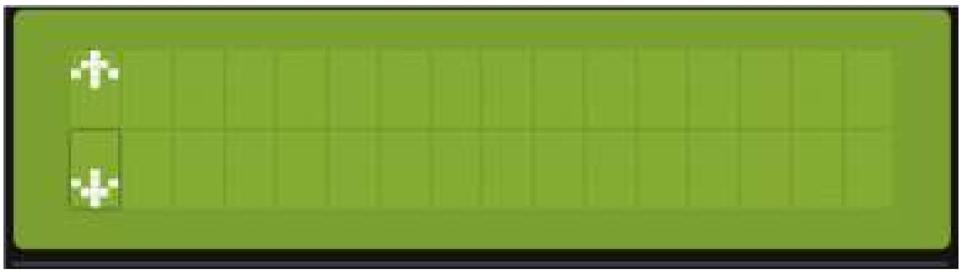
I used the above code to define 2 custom up and down arrows.



```
void setup() {
  lcd.begin(16, 2);
  lcd.createChar(0, upArrow);
  lcd.createChar(1, downArrow);
```

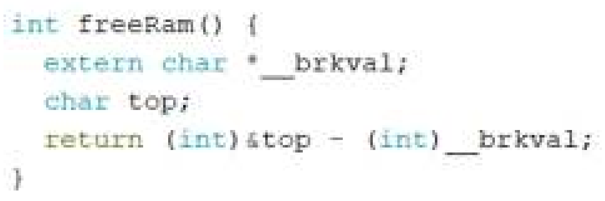
In my setup function I used the createChar function to apply my custom arrows to a number. In order to print the arrows I used the lcd.write() function.

My arrows ended up looking like this:



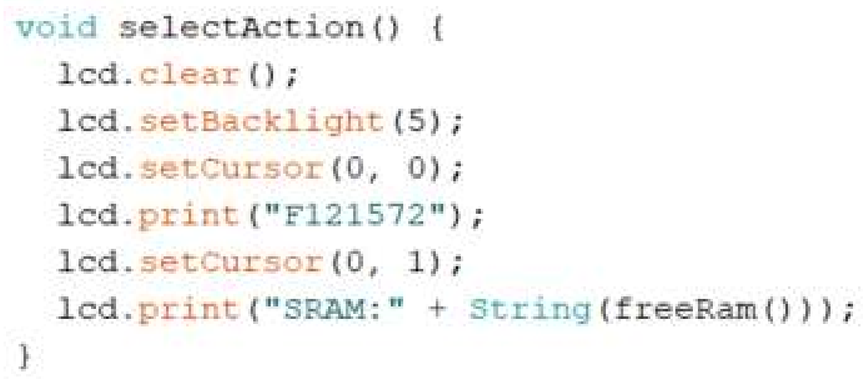
6 FREERAM

I used the following code to return the amount of free SRAM (taken from the lab 3 worksheet).



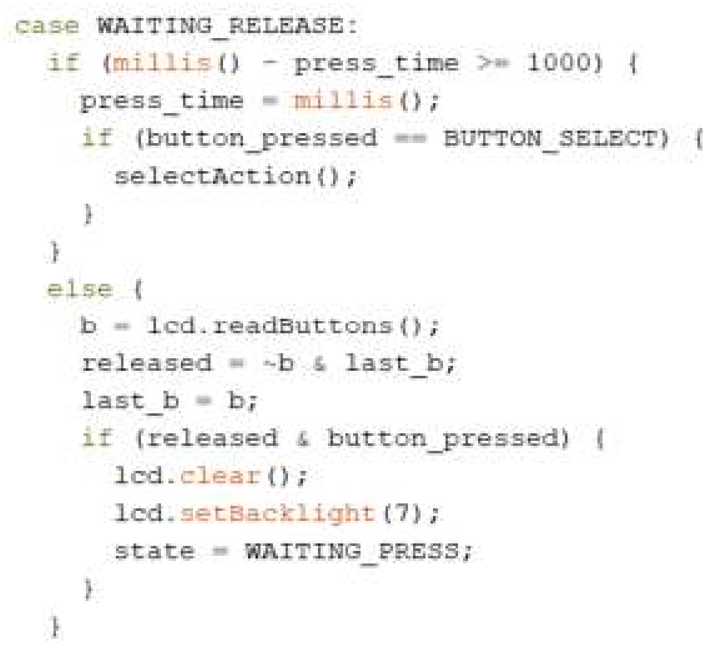
```
int freeRam() {
  extern char *__brkval;
  char top;
  return (int)&top - (int)__brkval;
}
```

I then used the following code to display the free SRAM:



```
void selectAction() {
  lcd.clear();
  lcd.setBacklight(5);
  lcd.setCursor(0, 0);
  lcd.print("F121572");
  lcd.setCursor(0, 1);
  lcd.print("SRAM:" + String(freeRam()));
}
```

This function would be called in the finite state machine when the select button had been pressed for more than 1 second.



```
case WAITING_RELEASE:
  if (millis() - press_time >= 1000) {
    press_time = millis();
    if (button_pressed == BUTTON_SELECT) {
      selectAction();
    }
  }
  else {
    b = lcd.readButtons();
    released = ~b & last_b;
    last_b = b;
    if (released & button_pressed) {
      lcd.clear();
      lcd.setBacklight(7);
      state = WAITING_PRESS;
    }
  }
```

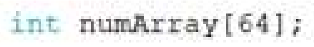
7 HCI

I was not able to complete this extension.

8 EEPROM

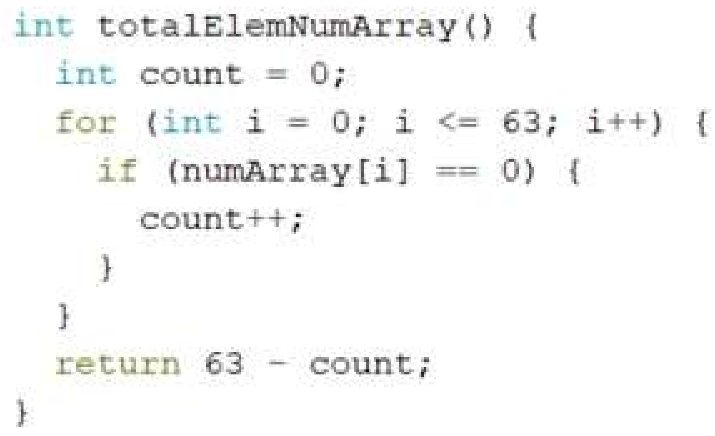
I was not able to complete this extension.

9 RECENT



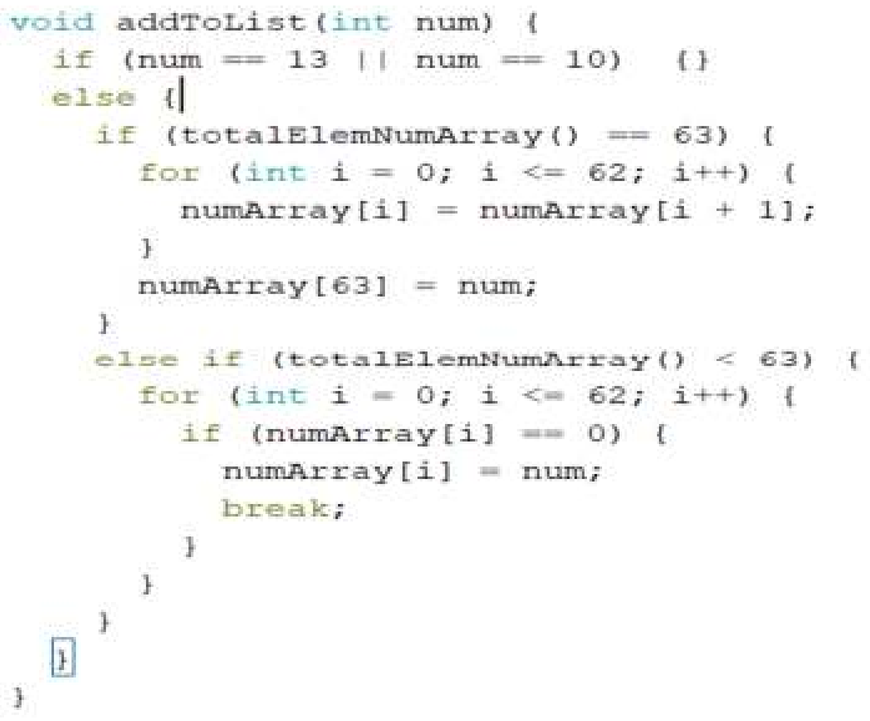
```
int numArray[64];
```

First I declared an array to store 64 numbers.



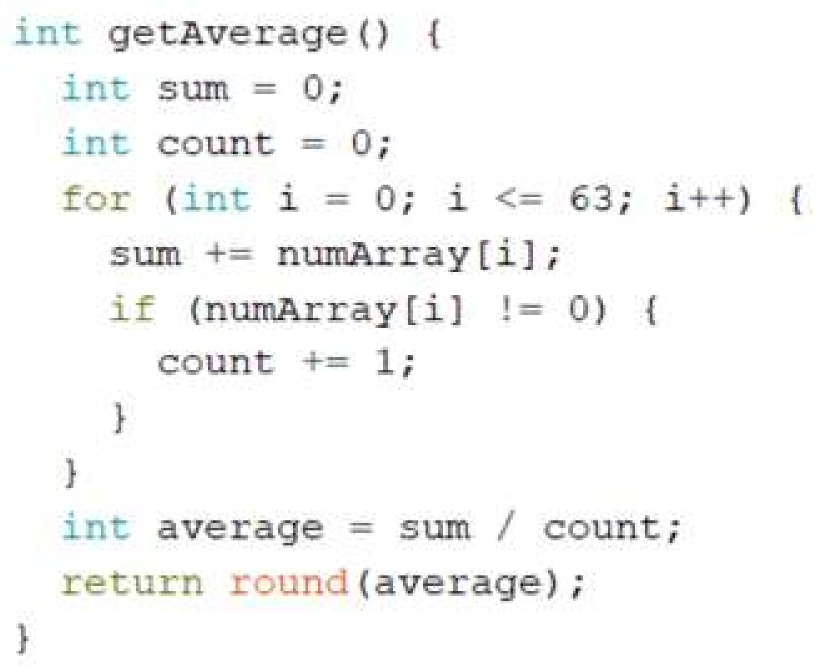
```
int totalElemNumArray() {
  int count = 0;
  for (int i = 0; i <= 63; i++) {
    if (numArray[i] == 0) {
      count++;
    }
  }
  return 63 - count;
}
```

I also created the above function that would count the number of empty elements in the array (since by default the array's values are set to 0) and would return the total number of elements that are occupied in the array.



```
void addToList(int num) {
  if (num == 13 || num == 10)   {}
  else {
    if (totalElemNumArray() == 63) {
      for (int i = 0; i <= 62; i++) {
        numArray[i] = numArray[i + 1];
      }
      numArray[63] = num;
    }
    else if (totalElemNumArray() < 63) {
      for (int i = 0; i <= 62; i++) {
        if (numArray[i] == 0) {
          numArray[i] = num;
          break;
        }
      }
    }
  }
}
```

The above function checks if every element in the array has been written to. If so, it shifts the values in the array left once and appends the new number to the end of the list, making the array act like a queue. If not every element in the array has been written to, it goes through the list and adds the new number to the first empty spot it finds. This function ensures that only the most recent 64 values entered into the serial monitor are considered.

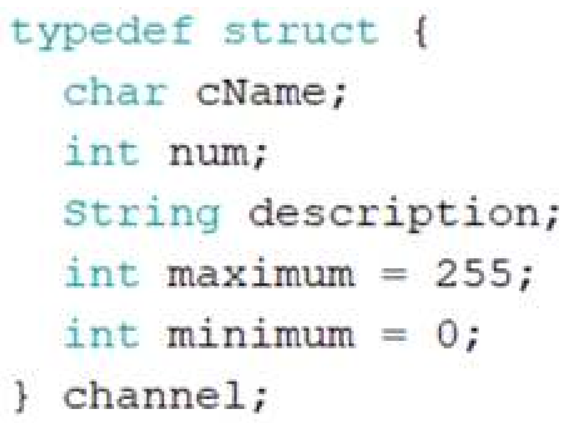


```
int getAverage() {
  int sum = 0;
  int count = 0;
  for (int i = 0; i <= 63; i++) {
    sum += numArray[i];
    if (numArray[i] != 0) {
      count += 1;
    }
  }
  int average = sum / count;
  return round(average);
}
```

The final function for this extension calculates the average of all of the elements in the numArray. It does this by adding every element to the variable sum, and whenever the element at the desired index is not empty it increments the count variable by 1. The average is then given by the sum/count. The value is rounded to the nearest integer by using the round() function. The average is then displayed on the LCD alongside the other pieces of data.

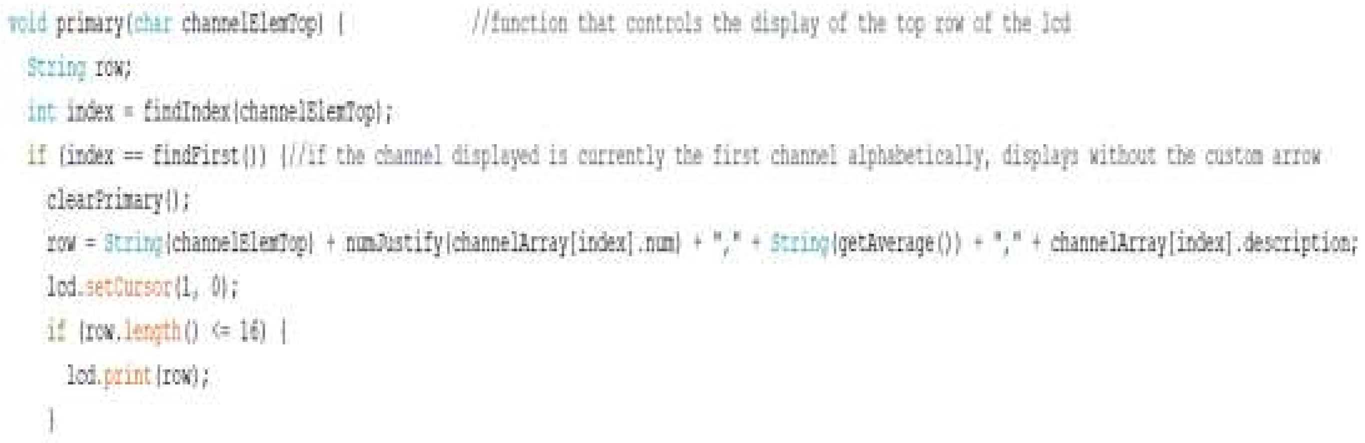
10 NAMES

In order to store the channel name(description) I used a String variable:



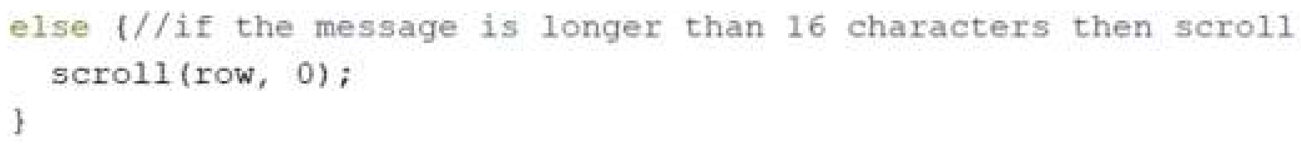
```
typedef struct {
  char cName;
  int num;
  String description;
  int maximum = 255;
  int minimum = 0;
} channel;
```

The description would then be added to a String variable (row) that would contain all of the information regarding a channel:



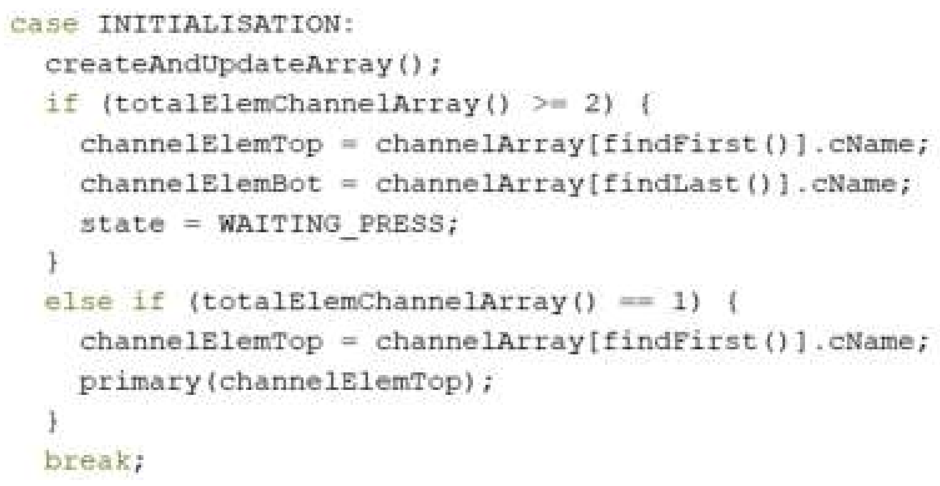
```
void primary(char channelElemTop) {            //function that controls the display of the top row of the lcd
  String row;
  int index = findIndex(channelElemTop);
  if (index == findFirst()) {//if the channel displayed is currently the first channel alphabetically, displays without the custom arrow
    clearPrimary();
    row = String(channelElemTop) + numJustify(channelArray[index].num) + "," + String(getAverage()) + "," + channelArray[index].description;
    lcd.setCursor(1, 0);
    if (row.length() <= 16) {
      lcd.print(row);
    }
```

If the length of row was more than 16 characters then it would apply the scroll function.

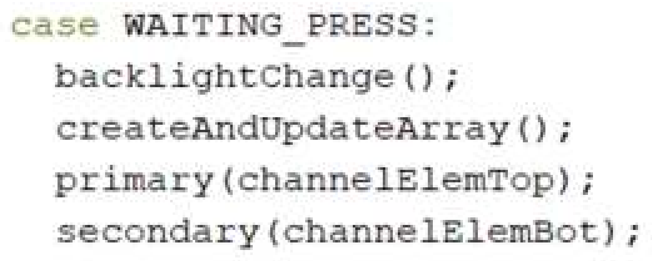


```
else {//if the message is longer than 16 characters then scroll
  scroll(row, 0);
}
```

I called the primary() and secondary() functions in my FSM whenever I needed to update the display, as shown below:



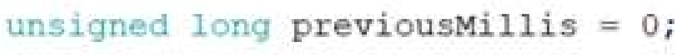
```
case INITIALISATION:
  createAndUpdateArray();
  if (totalElemChannelArray() >= 2) {
    channelElemTop = channelArray[findFirst()].cName;
    channelElemBot = channelArray[findLast()].cName;
    state = WAITING_PRESS;
  }
  else if (totalElemChannelArray() == 1) {
    channelElemTop = channelArray[findFirst()].cName;
    primary(channelElemTop);
  }
  break;
```



```
case WAITING_PRESS:
  backlightChange();
  createAndUpdateArray();
  primary(channelElemTop);
  secondary(channelElemBot);
```

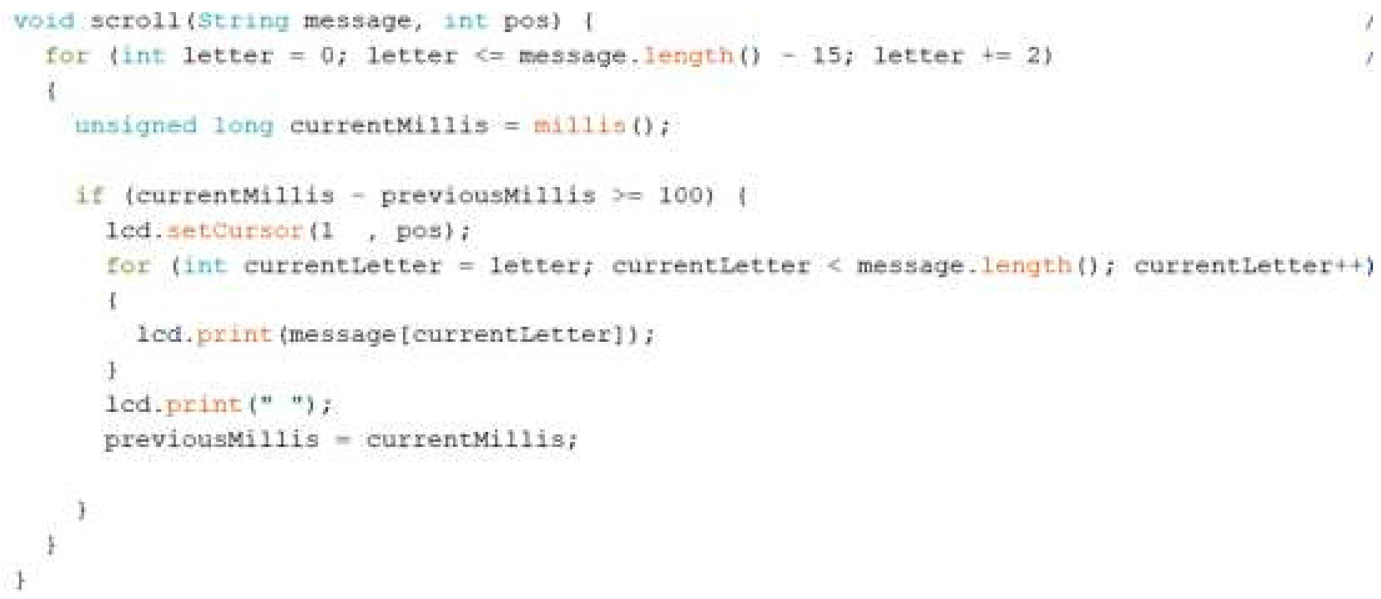
If I were to change anything about my code, I would change the fact that I stored the description variable as a string, which is not as memory efficient as an array of chars. However, my program did not struggle with memory management, so I was able to utilise strings and their associated functions, such as substring().

11 SCROLL



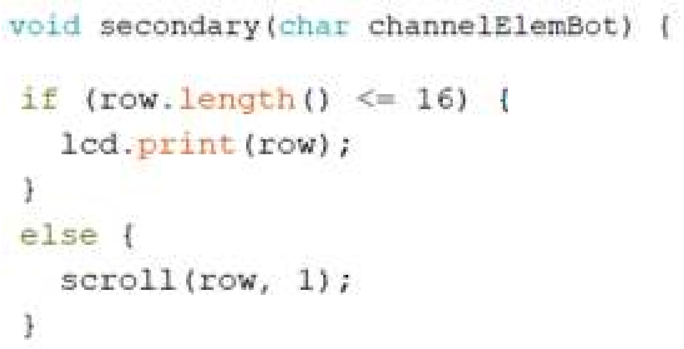
```
unsigned long previousMillis = 0;
```

First I had to create a variable called previousMillis that would store the last time that the for-loop had finished, which would be compared to the current time in order to determine if 1000ms had passed.



```
void scroll(String message, int pos) {                                          /
  for (int letter = 0; letter <= message.length() - 15; letter += 2)          /
  {
    unsigned long currentMillis = millis();

    if (currentMillis - previousMillis >= 100) {
      lcd.setCursor(1  , pos);
      for (int currentLetter = letter; currentLetter < message.length(); currentLetter++)
      {
        lcd.print(message[currentLetter]);
      }
      lcd.print(" ");
      previousMillis = currentMillis;

    }
  }
}
```

Then I created the above function which works by cycling through all of the letters of the input message and printing them with a delay of 100 ms, and then repeating it for the rest of the LCD row.



```
void secondary(char channelElemBot) {

 if (row.length() <= 16) {
   lcd.print(row);
 }
 else {
   scroll(row, 1);
 }
```

In the primary() and secondary() functions, it checks if the length of the message (row) is greater than 16. If it is, then it prints the message using the scroll() function.

I was unfortunately not able to make the program scroll at 2 characters per 1000ms, because the scrolling would simply not work or the message would not even appear on the LCD. Looking back, I think that perhaps the problem was caused by the timing interval not registering properly, since at intervals larger than 100ms it would not even register a scroll happening or it would take much longer than 1000ms to scroll in random steps. This is the main issue in my code currently. If I were to try to fix it again, I would try to implement a completely different FSM that could perhaps include another method to scroll.