Main aim of this project is to create and interface between LCD, Keypad and TM4C123GH6PM to design a calculator.  
  
During week 7 initial project work started. This includes going through project requirements, datasheets for different devices and thinking about potential features and functions to include.

To successfully complete this project I have identified key objectives:  
Correctly connect LCD and Keypad to the microcontroller

Initialise and interface with Keypad

Initialise and interface with LCD, display keypad inputs

Perform simple calculations

Implement floating-point and nested calculations

Implement password access

\*More additional features will be considered. This depends on how fast I can write essential functions first and have an initial working version of the calculator.

Main functions that I though about so far that will be needed: initialise components, get input from keypad, send output to LCD, conversions between string/int/char, calculations. This list fill expand during the implementation.

A picture containing box and whisker chart

Description automatically generated

Week 8

In week 8 I spent my time writing keypad, LCD, timer functions. This includes initialization and interfacing. All the functions needed were written and tested.

To test my code, I’ve used new functions in main.c:

* To test Keypad and LCD - Press key and display result (see KeypadLCDTest.png)
* To test LCD functions (display strings, move cursors, clear screen) – Print string in line one, move cursor, print string in line 2 (see LCDstringTest.png)

Main issues that I’ve encountered:

* readKeypad() everything worked fine apart from function returning ‘1’ instead of ‘A’ when A is pressed. This was fixed by adding small delay after writing to PORTD bits.
* LCD displaying weird symbols. I’ve made a mistake and send byte data incorrectly, however, after looking at datasheet again I’ve fixed it by shifting first nibble.  
   (**LCD\_DB = (DB&0xF0) >>4)** instead of **LCD\_DB = (DB&0xF0) >>4)**

Hardware schematic did not change and can be found in Week 7 entry (see [Connections.png](https://minerva.leeds.ac.uk/courses/1/202122_30389_ELEC3662/blog/_114903_1/post/_287418_1/Connections.png)).

Initial flowchart outlining main functionality was created using draw.io. (see Flowchart.png). My plan is to read values into char array and after ‘=’ is pressed calculate the result. For this scanning and parsing algorithms will be used. This is subject to change during implementation as a new approach may be chosen in later stages.

Week 9

In week 9 I have created routine that process inputs coming from keypad and concatenates math symbols into one char array (math\_exp) which will later be evaluated.

My input routine can handle:

* + - Numbers
    - Symbols + - x / . E ( )
    - Shift/Unshift function
    - Rubout last character
    - Delete entire entry
    - End input

First my input routine reads keypad value checks whether ‘\*’ was pressed which means end of input. Then checks if shift flag needs to be changed if not it process input and returns math symbol or ‘R’ (rubout), ‘Z’ (delete entry). math\_exp char array is thn updated and result is displayed.

Please see Input.png as it contains flow chart and input processing explanation (i.e., pressed button meaning).

Functionality have been tested by printing various values to LCD. (see LCDInput.png). End input/rubout last/delete entry were tested as well and work fine.

https://en.wikipedia.org/wiki/Recursive\_descent\_parser  
 Week 10

After week 9 I now had a mathematical expression stored in char array. Therefore, in week 10 I spent a significant amount of time researching various techniques on how this string could be processed and evaluated. To achieve this some kind of parsing algorithm needs to be used. For my application I have decided to use a Recursive Descent Parser (https://en.wikipedia.org/wiki/Recursive\_descent\_parser) .

My Recursive Descent parser:

* Predictive parser.
* It will use LL(1) grammar.
  + Input is read from left to right.
  + Produces left-to-right derivation.
  + Uses 1 lookahead token.
* Parser will need to handle inputs ‘+’, ‘-’, ‘x’, ‘/’, ‘(’, ‘)’, ‘^’, ‘E’, ‘[0..9]’.
* Grammar that will be used for my purposes (I am referencing to the examples provided in <https://en.wikipedia.org/wiki/Recursive_descent_parser> and http://teaching.idallen.com/cst8152/98w/recursive\_decent\_parsing.html)  
  **expression = ["+"|"-"] term {("+"|"-") term}  
  term = factor {(“\*” | ”/” | “^” | “E”)factor}  
  factor = unsigned\_factor | -unsigned\_factor  
  unsigned\_factor = number | (expression)**

The implementation of the recursive descent parser is relatively simple. I will need a buffer to hold token values, a global variable that gives me current symbol from math\_exp array and look ahead token values will be defined by enumeration. It will need a scanner to identify token value from input. Then evaluation functions from grammar above will be written: float expression(), float term(), float factor(), float unsigned\_factor(). See (FuntionsVariables.png).

As this program does not require interfacing with LCD and Keypad I am planning to create separate C file and use Code::Blocks IDE. This will simplify testing as printing values will be easier. After I have working program I will copy it over into my project.

Week 10.

I was implementing mathematical expression which is stored in char array evaluation. For this I am using recursive descent parser approach.

Functions implemented (see pictures attached for code).

* void read() – puts current symbol into token buffer and reads next symbol from math\_exp array.
* void reset() – clear token buffer array
* int scan() – scans current symbol and returns token type which was defined in look\_ahead enumeration type. If current symbol is a number this function will continue reading into token buffer array until non number character is reached. This will later on be used to retrieve float number using sscanf().
* void next() – update look\_ahead token.
* void initialise\_parser() – initializes parser. Resets token buffer, gets first symbol from math\_exp, and scans it.
* Following functions are recursive descent parsing and evaluation and follows grammar stated in last weeks journal entry:
  + float unsigned\_factor()
  + float factor()
  + float pow\_factor()
  + float term()
  + float expression()
* I am using static int global variable to check for errors.

After this I have fully functioning calculator. Next week I will tidy up my main function. The routine for the calculator will be written.

Additional entry – Week C1 (Christmas holiday)

This week I have updated calculator input routine and added password protection.  
  
Input routine updates:

* New function SHIFT/4 – gets previous answer and add to mathematical expression.
* Previously I was clearing LCD and printing whole math expression every time new symbol was entered. I have changed this to clear LCD and reprint expression only when character/whole string is deleted.
* Improved input functionality. Previously if the key was pressed the program would write it continuously with short delay after each entry. This caused some problems i.e., multiple entries, next button not registered because of delay. I have changed this, so a button needs to be released before printing new one. This improved user experience significantly.

Answer printing:

* Answer is printed on the second line and is always kept there until new expression is evaluated. This resembles real calculator as you can see previous answer while entering new expression. (see Answer.png)

Password protection: (Password.png)

* Added 4 digits password.
* Password is stored in flash memory (address = 0x00008000).
* To implement this functionality, I have looked at datasheets and found relevant information (see Flashdatasheet.png). However, to avoid errors I have taken Flash programming code from a book and edited it for my purposes. Book used:
  + "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",

ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2016

* + "Embedded Systems: Real-Time Operating Systems for ARM Cortex-M Microcontrollers",

ISBN: 978-1466468863, Jonathan Valvano, copyright (c) 2016

* Password routine:
  + Entry password: -> If wrong prompt to enter again / If correct move on -> Ask to press A to change password -> If A is pressed go into password change routine / If any other key is pressed move on.

Right now, my calculator is finished. Maybe I’ll add functionality to display graphics.