Serial Communication Calculator

Due Date: Dec 15, 2014

Department of Electrical Engineering
University of South Florida

EEL 4743L – MICROPROCESSORS/EMBEDDED SYSTEMS LABORATORY
Fall 2014

Team Members:

Miguel A. Naranjo U50136978

Juan C. Morales U32773102

Abstract:

The Serial Communication Calculator is a serial communication interface based calculator. By the means of serial communication software such as **TeraTerm**, the user is prompted to enter a two numbers with a calculation operation between them (“+”, “-”, “*”, “/”, “^”). The numbers entered can be either **Integers** or **Float** numbers, with or without decimal places. Note that the software only takes numbers and gives an **ERROR** messages if a character other than **ENTER** or **BACKSPACE** are entered. **ENTER** is understood as the key needed for the program to provide the result of the mathematical calculation and **BACKSPCAE** is understood as the key to erase the last entry to be replaces for a new entry.

After data has been entered and validated and data is stored in the microcontroller, the board (FRDMKL25Z) performs the mathematical calculation and prints the result to the screen via serial communication, and also prints it to the LCD display.

Is there are any errors during the process of saving the needed information of allocating memory, the LCD display as well as the serial monitor will show an ARROR message.

Project Objectives/Goals:

The idea of this project came about having to use the serial communication interface in on the Lab experiments. Using a similar approach to that experiment, we decided to implement a fully functional calculator that is capable of Addition, Subtraction, Multiplication, Division, and Power.

As a regular calculator, such as *Fx-115ES PLUS*, we implemented code and used libraries such as **TextLCD.h**, so that the result of the calculation was displayed on the display as well as on the TeraTerm terminal.

The written software can handle decimal numbers as well as natural numbers, but not a combination of both. **E.g. 1.23 + 12**. Never the less, if the natural number is typed as: **12.0** instead of **12**, the software can normally handle it.

Using C to write the code was one of the goal of this project as well, since that gave us a chance to get a better understanding of more complex programming.

Background Knowledge:

For the thorough understanding of this project we need to be familiar with the basics of C programming. One of the most basic skills when programming in c is knowing who to use the *if()* statement. As it will be noticed through the code, this is used with quite frequently. The use of *switch()* statements, the use of *pointers*, the use of *structures*, the use of *for() loops*, the use of *while() loops*, the use of memory allocation using *calloc()*, and the multiple ways of printing information to the display and getting information from it, *printf()* and *csanf()* accordingly, are necessary to understand the basic flow of the code.

It is also of great importance the understanding of functions; the way they work and the way addresses of values are passes to the. Knowing how pointers work and how elements from a structure are accessed from a functions are basic for the understanding of the code. Also it is imperative to know when and when a function needs to returns a value, and what is done with the value if is either is returned or not.

In writing this program, we used the HEX values of some the key on the keyboard to reference the typed numbers or letters. E.g. **ENTER = 0X0D**. The knowledge of the equivalent for all the numbers on the keyboard and the equivalent for all the mathematical operations, need to be known and understood.

Since we are working with a serial terminal emulator, is good to know the reason we have so that when there is a print statement, we print a *\n (new line)* and a *\r (carriage return).* Normally, in a windows application, the *\n* is enough. In this application the *\n* only makes the printer jump to the next line, and does not make it return to the origin of the line. This causes the next printed statement start printing right where the last printed statement finished, just one line below. The need for *\r* come because every time we print something to the **TeraTerm** terminal, we want it to print at the beginning of the line.

It is necessary to understand how a voltage divider works and how a potentiometer has its basics on a voltage divider. Understanding the amount of current, through an LED will help in grasping the concept of being able to deem the LCD characters' brightness.

Required Software/Hardware:

For the implementation of this calculator, the following items were needed:

1. Breadboard
2. 14 male to male jumpers
3. 5kΩ potentiometer
4. FRDKL25Z board
5. ADM1602K 16x2 LCD Display
6. USB to Mini-USB connector

There is no software needed, because in this project an online compiler is used, then the binary file that goes programmed in the microcontroller, is just downloaded onto the board. In order to be able to successfully connect the microcontroller to the PC, the P&E OpenSDA USB Driver is needed.

The ADM1602K 16x2 LCD Display is a display that comes with two rows of 16 small boxes, each of which, can display the character sent to it by the program.

Having these items is just part of what needs to be done. Here we used the 5VDC and the GND provided to the board to power up the backlight of the LCD display, which will be explained in more detail in the next section. Also, the brightness of each of the individual squares of the 16x2 LCD display was changed by using the 5kΩ potentiometer.

The board used did not come with the plastic extension to connect the jumpers, so we had to solder those in place in order to be able to connect to the LCD. Accordingly, the LCD got its pins (16) soldered onto it.

Design Approach/Procedure:

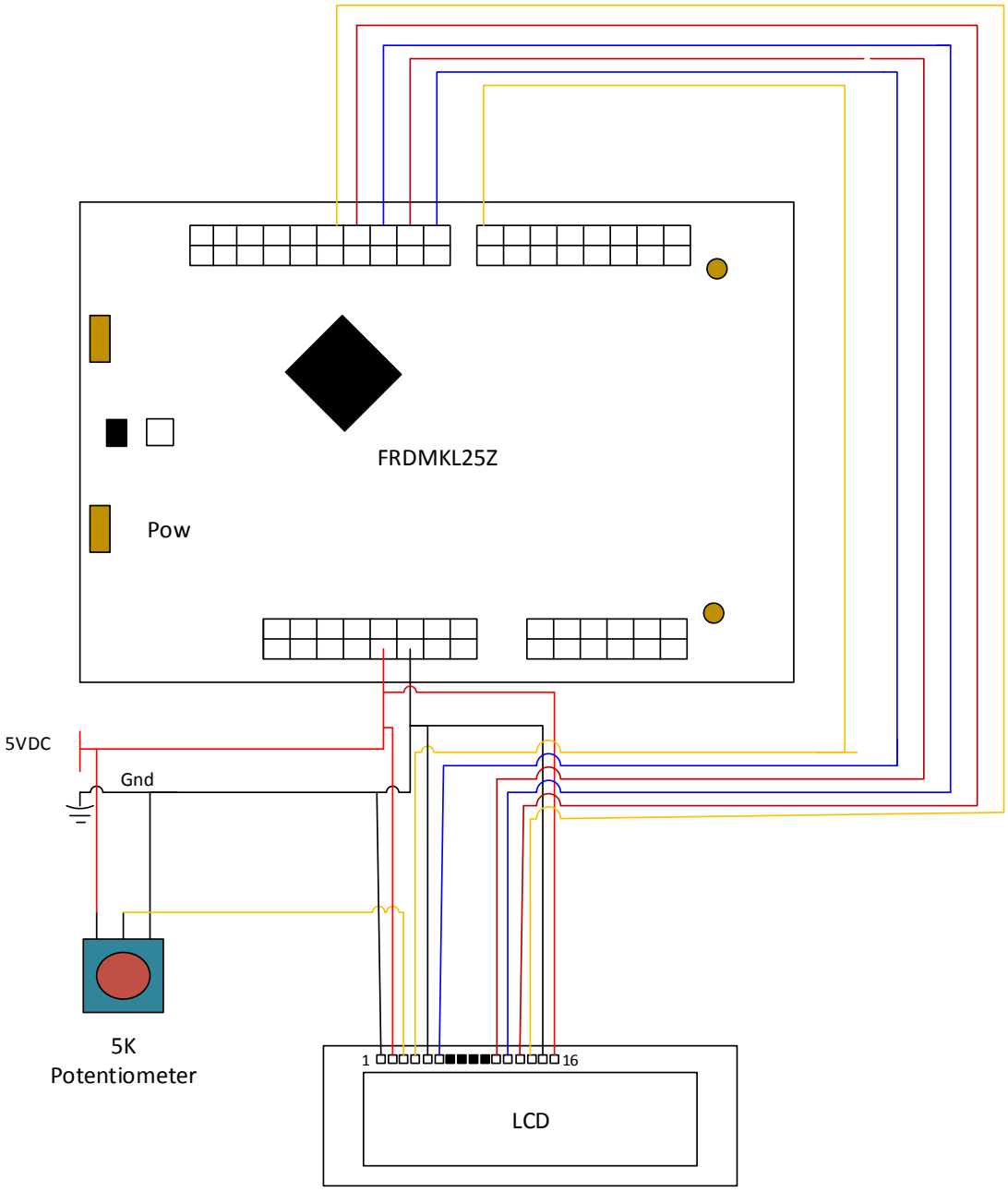
As it can be seen in the following diagram, the LCD display is connected to both, ground (GND) and 5VDC (Vcc), provided by the microcontroller. These GND and 5VDC, (pin 15 and pin 16) respectively, lit the backlight of the display.

Pin 3 of the LCD is provided with a voltage, from the voltage divider (potentiometer), thus allowing the brightness of the displayed number be more or less.

Two of the legs of the potentiometer are connected to 5VDC and GND (1^{st} and 3^{rd} legs), and the middle one (2^{nd}) provide the voltage for the previously mentioned brightness of the numbers.

The I/O pins PTC9, PTA13, PTD5, PTD0, PTD2, and PTD3 of the microcontroller are used to send the data serially to the display. The LCD library consequently handles these pins and provides the correct data to each of the pins to the LCD every time the desired character is sent. This library uses the choses I/O pins to multiplex.

The following figure shows the connections interface between the display and the RD-KL25Z.


FRDMKL25Z
Pow
5VDC
Gnd
5K
Potentiometer
1
16
LCD

Design Code:

The written code works in the following manner:

After setting the baud rate in the TeraTerm, so that it matches the baud rate of the program, for serial communications, the **RESET** button of the board is pressed to begin.

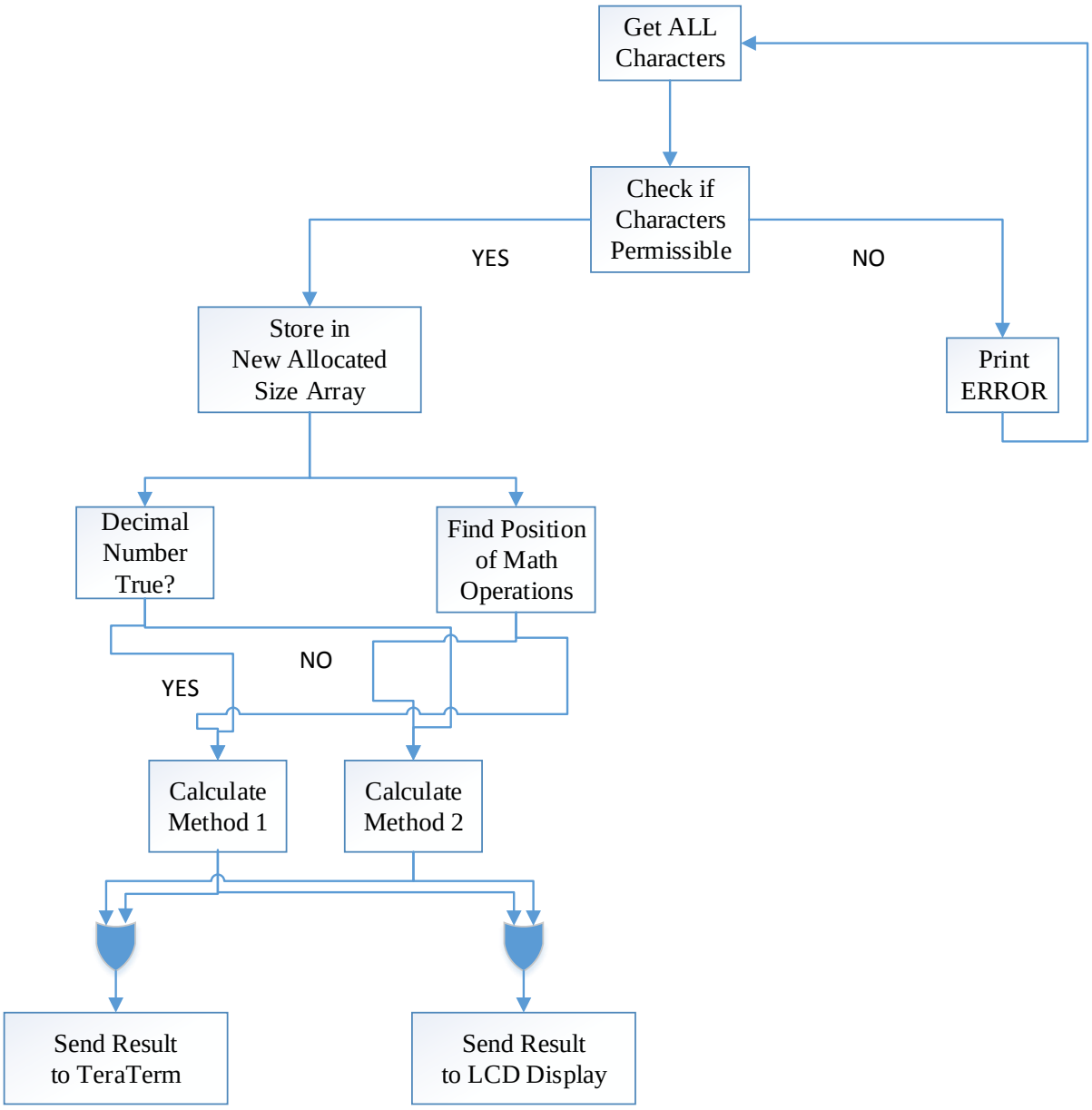
When the program begins, it prompts the used to enter numbers, separated by ma mathematical operation (“+”, “-”, “*”, “/”, “^”). if a character other than the desired characters (math operators or numbers) is entered, the program will give an **ERROR** and prompt the user to enter the numbers again.

Once the **ENTER** key is pressed, after the numbers with the math operation is entered, the program will begin to which the array into which the all the characters were saved. It will inspect each element of the array. *Remember that the size of this array was allocated base on the number of correct characters entered. If for some reason memory cannot be allocated, the program will give an ERROR message.* When inspecting, the program will determine the location od the mathematical operation, and will also determine the position of the decimal point (if any) in both numbers, thus finding its location if it exists.

After the position of the math operation and the decimal point (if any) is known, the program will begin to perform the desires math operation specified by the user. Since the program can handle decimal and natural numbers, is accounts with two ways of finding the numbers with which it will perform the operation. One way, it will notice that there is no decimal point, and the other, it will account for the decimal point.

Finally after the calculation is performed, the program will display the result to TeraTerm and to the LCD display. At this point both results must match.

Next page contains a block diagram of the flow of the program for easier understanding.


Get ALL Characters
Check if Characters Permissible
YES
NO
Store in New Allocated Size Array
Print ERROR
Decimal Number True?
Find Position of Math Operations
NO
YES
Calculate Method 1
Calculate Method 2
Send Result to TeraTerm
Send Result to LCD Display

Source Code: (main.cpp)

```
/*This program was created to be used with TeraTerm terminal reader on
Host PC */

#include "mbed.h"
#include <stdint.h>
#include <stdio.h>
#include "math.h"
//#include "ACM1602.h"
#include "TextLCD.h"

//#define hundredsArray [] = {1000000, 100000, 10000, 1000, 100, 10, 1}
//#define oneOverHundreds [] = {0.0000001, 0.000001, 0.00001, 0.0001,
0.001, 0.01, 0.1}


#define bufferZise 2000
#define bufferZise_2 100

Serial pc(USBTX, USBRX);
TextLCD lcd(PTC9, PTA13, PTD5, PTD0, PTD2, PTD3); // rs, e, d4-d7
Timer t;

struct characterString
{
    unsigned char array[bufferZise];
//    float calculationArray[];

   int *integerValuesArray;  // move array to inside the structure
   int *calculationArray;    // move array to inside the atructure
   int positionOfAnyOperation;    // this will contain teh position on the
array of any of the math operatons

    float valSum;  // Sum  "+"
    float valSub;  // Substraction  "-"
    float valMul;  // Multiplication "*"
    float valDiv;  // Division "/"
    float valPow;  // Power "^"

    float valSum_before;  // Sum  "+"  ==> Number to the left of operation
    float valSub_before;  // Substraction  "-"  ==> Number to the left of
operation
    float valMul_before;  // Multiplication "*"  ==> Number to the left of
operation
    float valDiv_before;  // Division "/"  ==> Number to the left of
operation
    float valPow_before;  // Power "^"   ==> Number to the left of
operation

    float valSum_after;  // Sum  "+"  ==> Number to the right of operation
    float valSub_after;  // Substraction  "-"  ==> Number to the right of
operation
```

```
    float valMul_after;  // Multiplication "*"  ==> Number to the right of
operation
    float valDiv_after;  // Division "/"  ==> Number to the right of
operation
    float valPow_after;  // Power "^"   ==> Number to the right of
operation

    int mulBeforeMath;  //variables thet tell what to multiply by or
devide by to
    int divBeforeMath;  // get the numerical value of
access.calculationArray[]
    int mulAfterMath;
    int divAfterMath;

    bool sumFlag;     // Flag to detenct that the characer that represents
a math operation is in the calculationArray[]  "+"
    bool subFlag;     // Flag to detenct that the characer that represents
a math operation is in the calculationArray[]  "-"
    bool divFlag;     // Flag to detenct that the characer that represents
a math operation is in the calculationArray[]  "/"
    bool mulFlag;     // Flag to detenct that the characer that represents
a math operation is in the calculationArray[]  "*"
    bool powFlag;     // Flag to detenct that the characer that represents
a math operation is in the calculationArray[]  "^"
    bool decPlaceFlag;// Flag to detenct that the characer that represents
a math operation is in the calculationArray[]  "."

    bool firstDecimalPiont;  // tells if there is decimal point before
math operation
    bool secondDecimalPiont;  // tells if there is decimal point after
math operation

    int plusCharacterLocation;
    int minusCharacterLocation;
    int multiplicationCharacterLocation;
    int divisioCharacterLocation;
    int powerCharacterLocation;

    float valueBeforeDot_BeforeMath;
    float valueAfterDot_BeforeMath;
    float valueBeforeDot_AfterMath;
    float valueAfterDot_AfterMath;

    int programCounter;


//*****************************************************************************
*****************************************************
    //*********************************Variables used inside void
decPlacedFunction(characterString *ptr2, int k) **********************

    int posDot_1;  // counter for location of " . " before math operation
symbol
```

```
    int posDot_2;  // counter for location of " . " after  math operation
symbol


    int posDot_Before_Math, posDot_After_Math;    // these will have the
numerical location of where the "." is, AFTER  the mathematical operation
symbols:  ( * / + - ^ )

    int posSumOperation;  // position of mathematical operation ADD "+"
    int posSubOperation;  // position of mathematical operation DUB "-"
    int posMulOperation;  // position of mathematical operation MUL "*"
    int posDivOperation;  // position of mathematical operation DIV "/"
    int posPowOperation;  // position of mathematical operation POW "^"

//*****************************************************************************
**********************************************************

//*****************************************************************************
**********************************************************

}access;

void deleteFunction(characterString *ptr1)   // not in use
{
    access.plusCharacterLocation = 0;
    access.minusCharacterLocation = 0;
    access.multiplicationCharacterLocation = 0;
    access.divisioCharacterLocation = 0;
    access.powerCharacterLocation = 0;

   return;
}

void decimalPositionsFunction(characterString *ptr1)    // clears
access.(mul/div)(before/after)Math ()
{
    access.mulBeforeMath = 0;  //variables thet tell what to multiply by
or devide by to
    access.divBeforeMath = 0;  // get the numerical value of
access.calculationArray[]
    access.mulAfterMath = 0;
    access.divAfterMath = 0;
}

void clearFunction(characterString *ptr1)  // Function that will arase any
data previously used when program running
{
    for(int y = 0; y < bufferZise; y++)
    {
        access.array[y] = NULL;  //Ensure avery entry of Array is empty at
startup.
    }

    access.valSum = 0;  // final value or arithmetic opertions
```

```
    access.valDiv = 0;
    access.valMul = 0;
    access.valDiv = 0;
    access.valPow = 0;
    access.positionOfAnyOperation = 0;

    access.mulBeforeMath = 0;  //variables thet tell what to multiply by
or devide by to
    access.divBeforeMath = 0;  // get the numerical value of
access.calculationArray[]
    access.mulAfterMath = 0;
    access.divAfterMath = 0;

    access.valSum_before = 0;  // Sum  "+"  ==> Number to the left of
operation
    access.valSub_before = 0;  // Substraction  "-"  ==> Number to the
left of operation
    access.valMul_before = 0;  // Multiplication "*"  ==> Number to the
left of operation
    access.valDiv_before = 0;  // Division "/"  ==> Number to the left of
operation
    access.valPow_before = 0;  // Power "^"   ==> Number to the left of
operation

    access.valSum_after = 0;  // Sum  "+"  ==> Number to the right of
operation
    access.valSub_after = 0;  // Substraction  "-"  ==> Number to the
right of operation
    access.valMul_after = 0;  // Multiplication "*"  ==> Number to the
right of operation
    access.valDiv_after = 0;  // Division "/"  ==> Number to the right of
operation
    access.valPow_after = 0;  // Power "^"   ==> Number to the right of
operation

    access.sumFlag = false;
    access.subFlag = false;
    access.divFlag = false;
    access.mulFlag = false;
    access.powFlag = false;
    access.decPlaceFlag=false;

    access.posDot_1 = 0;  // counter for location of " . " before math
operation symbol
    access.posDot_2 = 0;  // counter for location of " . " after  math
operation symbol

    access.posDot_Before_Math = 0;
    access.posDot_After_Math = 0;

    access.posSumOperation = 0;  // position of mathematical operation ADD
"+"
    access.posSubOperation = 0;  // position of mathematical operation DUB
"-"
```

```
    access.posMulOperation = 0;  // position of mathematical operation MUL
"*"
    access.posDivOperation = 0;  // position of mathematical operation DIV
"/"
    access.posPowOperation = 0;  // position of mathematical operation POW
"^"

    access.valueBeforeDot_BeforeMath = 0;  // float Value of number before
dot and before math operator
    access.valueAfterDot_BeforeMath = 0;   // float Value of number after
dot and before math operator
    access.valueBeforeDot_AfterMath = 0;   // float Value of number before
dot and after math operator
    access.valueAfterDot_AfterMath = 0;    // float Value of number after
dot and after math operator

    access.firstDecimalPiont = false;  // tells if there is decimal point
before math operation
    access.secondDecimalPiont = false;  // tells if there is decimal point
after math operation

    return;
}

void findNumbers(characterString *ptr2, int k)
{
        float hundredsArray[] = {1, 10, 100, 1000, 10000, 100000, 1000000,
100000000, 1000000000, 10000000000, 100000000000};
        float oneOverHundreds[] = {0.1, 0.01, 0.001, 0.0001, 0.00001,
0.000001, 0.0000001, 0.00000001, 0.000000001, 0.000000001, 0.0000000001};

    if( access.firstDecimalPiont == true)  // find out decimal values
entries only
    {
//        float hundredsArray[] = {1, 10, 100, 1000, 10000, 100000,
1000000, 100000000, 1000000000, 10000000000, 100000000000};
//        float oneOverHundreds[] = {0.1, 0.01, 0.001, 0.0001, 0.00001,
0.000001, 0.0000001, 0.00000001, 0.000000001, 0.000000001, 0.0000000001};

         int s = 0; // used for counting with before math operation numner
         int j = 0;  // used for counting with before math operation
numner

         int f = 0;  // used for counting with after math operation numner
         int w = 0;  // used for counting with after math operation numner

         //**************************************determine float number
before operation**************************************
         for(s = 0, j = access.mulBeforeMath - 1; s <
access.posDot_Before_Math; s++, j--)
         {
            access.valueBeforeDot_BeforeMath +=
access.calculationArray[s]*hundredsArray[j];   // calcuation of the float
value of the numbers entered in the array before the "DOT"
```

```
        }
//         pc.printf("float Value of number before dot and before math
operator : \t %f \n\r", access.valueBeforeDot_BeforeMath);


//         pc.printf("Position of Opertion %d
\t\n\r\n\r",access.positionOfAnyOperation);

        for(s = access.posDot_Before_Math + 1, j = oneOverHundreds[0]; s
< access.positionOfAnyOperation; s++, j++)
        {
           access.valueAfterDot_BeforeMath +=
access.calculationArray[s]*oneOverHundreds[j];
        }
//         pc.printf("float Value of number before dot and before math
operator : \t %f \n\r", access.valueAfterDot_BeforeMath);


        access.valSum_before = access.valueAfterDot_BeforeMath +
access.valueBeforeDot_BeforeMath;
//         pc.printf("Number to add before OPERATION: \t %f \n\r\n\r\n\r",
access.valSum_before);
//         pc.printf("Position of Opertion %d
\t\n\r\n\r",access.positionOfAnyOperation);

        //**************************************END of determine float
number before operation*************************************

        //**************************************determine float number
after operation*************************************

        //for(w = access.positionOfAnyOperation + 1, f =
hundredsArray[0]; w < access.posDot_After_Math; w++,f++)
        for(w = access.posDot_After_Math - 1, f = hundredsArray[0]; w >
access.positionOfAnyOperation; w--,f++)
        {
             access.valueBeforeDot_AfterMath +=
access.calculationArray[w]*hundredsArray[f-1];
        }
//         pc.printf("float Value of number before dot and after math
operator : %f \t\n\r\n\r",access.valueBeforeDot_AfterMath);

        for(w = access.posDot_After_Math + 1, f = oneOverHundreds[0]; w <
k; w++,f++)
        {
             access.valueAfterDot_AfterMath +=
access.calculationArray[w]*oneOverHundreds[f];
        }
//         pc.printf("float Value of number before dot and after math
operator : %f \t\n\r\n\r",access.valueAfterDot_AfterMath);
        access.valSum_after = access.valueBeforeDot_AfterMath +
access.valueAfterDot_AfterMath;
//         pc.printf("Number to add after OPERATION: \t %f \n\r\n\r\n\r",
access.valSum_after);
```

```
         //**************************************END of determine float
number before operation*************************************

     }
     else//  if( access.firstDecimalPiont == false)  // find out integers
entries only
     {
        int r = 0;
        int g =  0;

        for( g = 0, r = access.positionOfAnyOperation - 1; g <
access.positionOfAnyOperation; g++, r--)
        {
            access.valSum_before +=
access.calculationArray[g]*hundredsArray[r];
        }
//        pc.printf("float Value of number before math operator : \t %f
\n\r", access.valSum_before);

        for( g = access.positionOfAnyOperation + 1, r = (k -
access.positionOfAnyOperation - 1) ; g < k; g++, r--)
        {
            access.valSum_after +=
access.calculationArray[g]*hundredsArray[r - 1];
        }
//        pc.printf("float Value of number after math operator : \t %f
\n\r", access.valSum_after);

     }

     return;
}
void sumFunction(characterString *ptr2, int k)
{
    lcd.cls();

    access.valSum = access.valSum_after +  access.valSum_before;
//    lcd.printf("%f + %f = %f \n", access.valSum_before,
access.valSum_after, access.valSum);
    pc.printf("%f + %f = %f \n\r", access.valSum_before,
access.valSum_after, access.valSum);

    lcd.printf("%f",access.valSum);

    return;
}

void subFunction(characterString *ptr2, int k)
{
   lcd.cls();
   access.valSub = access.valSum_before -  access.valSum_after;
```

```
//   pc.printf("SUB OF TWO NUMBERS: \t %f \n\r\n\r\n\r", access.valSub);
    pc.printf("%f - %f = %f \n\r", access.valSum_before,
access.valSum_after, access.valSub);

   lcd.printf("%f",access.valSub);
   return;
}

void mulFunction(characterString *ptr, int k)
{
    lcd.cls();

    access.valMul = access.valSum_before*access.valSum_after;
//    pc.printf("Mul OF TWO NUMBERS: \t %f \n\r\n\r\n\r", access.valMul);
    pc.printf("%f * %f = %f \n\r", access.valSum_before,
access.valSum_after, access.valMul);

    lcd.printf("%f",access.valMul);
    return;
}

void divFunction(characterString *ptr2, int k)
{
    lcd.cls();

    access.valDiv = access.valSum_before/access.valSum_after;
//    pc.printf("Div OF TWO NUMBERS: \t %f \n\r\n\r\n\r", access.valDiv);
    pc.printf("%f / %f = %f \n\r", access.valSum_before,
access.valSum_after, access.valDiv);

    lcd.printf("%f",access.valDiv);
    return;
}

void powFunction(characterString *ptr2, int k)
{
    lcd.cls();
    access.valPow = pow(access.valSum_before,access.valSum_after);
//    pc.printf("Pow OF TWO NUMBERS: \t %f \n\r\n\r\n\r", access.valPow);
    pc.printf("%f ^ %f = %f \n\r", access.valSum_before,
access.valSum_after, access.valPow);

    lcd.printf("%f",access.valPow);
    return;
}

void decPlacedFunction(characterString *ptr2, int k)
{
    for(int i = 0; i <= k; i++)
    {
        if(access.calculationArray[i] == 88)         // Set position of
"+" sign in calculationArray[]
        {
            access.posSumOperation = i;
```

```
//            pc.printf("Position of Addition: %d\n\r",
access.posSumOperation);
            access.positionOfAnyOperation = i;
        }
        else if(access.calculationArray[i] == 77)    // Set position of "-
" sign in calculationArray[]
        {
            access.posSubOperation = i;
//            pc.printf("Position of Substraction: %d\n\r",
access.posSubOperation);
            access.positionOfAnyOperation = i;
        }
        else if(access.calculationArray[i] == 99)    // Set position of
"*" sign in calculationArray[]
        {
            access.posMulOperation = i;
//            pc.printf("Position of Multiplication: %d\n\r",
access.posMulOperation);
            access.positionOfAnyOperation = i;
        }
        else if(access.calculationArray[i] == 55)    // Set position of
"/" sign in calculationArray[]
        {
            access.posDivOperation = i;
//            pc.printf("Position of Division: %d\n\r",
access.posDivOperation);
            access.positionOfAnyOperation = i;
        }
        else if(access.calculationArray[i] == 44)    // Set position of
"^" sign in calculationArray[]
        {
            access.posPowOperation = i;
//            pc.printf("Position of Power: %d\n\r",
access.posPowOperation);
            access.positionOfAnyOperation = i;
        }

        else
        {
//            pc.printf("No Mathematical Opertion entered!!!\n\r");
//              access.positionOfAnyOperation = 0;
        }
    }
    pc.printf("\n\r");

        if (access.sumFlag == true)     // use the position on
posSumOperation() to find the position of the "." before the operator and
after the operator "+"
        {
            for(int dotBeroreSum = 0; dotBeroreSum <
access.posSumOperation; dotBeroreSum++)
            {
                if(access.calculationArray[dotBeroreSum] == 66)
                {
```

```
                    access.posDot_Before_Math = dotBeroreSum;
                    access.firstDecimalPiont = true;
                    access.mulBeforeMath = access.posDot_Before_Math;  //
number of entries between first entry and Dot
//                    pc.printf("\t\t%d\n\r",access.mulBeforeMath);

                    access.divBeforeMath = access.posSumOperation -
access.posDot_Before_Math - 1;  // number of entries between Dot and Sum
Position
//                    pc.printf("\t\t%d\n\r",access.divBeforeMath);

//                    pc.printf("Dot detected before sum at: %d\n
\r",access.posDot_Before_Math);
                }
                //pc.printf(" when no dot present: %d",
access.posDot_Before_Math );
            }
            pc.printf("\n\r");
            for (int dotAfterSum = access.posSumOperation + 1; dotAfterSum
<= k; dotAfterSum++)
            {
                if(access.calculationArray[dotAfterSum] == 66)
                {
                    access.posDot_After_Math = dotAfterSum;
                    access.secondDecimalPiont = true;
                    access.mulAfterMath = (access.posDot_After_Math -
access.posSumOperation - 1);
//                    pc.printf("\t\t%d\n\r",access.mulAfterMath);

                    access.divAfterMath = k - access.posDot_After_Math -1;
// number of entries between Dot and Sum Position
//                    pc.printf("\t\t%d\n\r",access.divAfterMath);

//                    pc.printf("Dot detected After sum at: %d\n
\r",access.posDot_After_Math);

                }
            }
            pc.printf("\n\r");
            access.sumFlag = false;
        }
//        pc.printf(" when no dot present: %d", access.mulBeforeMath );

        if (access.subFlag == true)     // use the position on
posSumOperation() to find the position of the "." before the operator and
after the operator "-"
        {
            for(int dotBeroreSub = 0; dotBeroreSub <
access.posSubOperation; dotBeroreSub++)
            {
                if(access.calculationArray[dotBeroreSub] == 66)
                {
                    access.posDot_Before_Math = dotBeroreSub;
                    access.firstDecimalPiont = true;
```

```
                    access.mulBeforeMath = access.posDot_Before_Math;  //
number of entries between first entry and Dot
//                    pc.printf("\t\t%d\n\r",access.mulBeforeMath);

                    access.divBeforeMath = access.posSubOperation -
access.posDot_Before_Math - 1;  // number of entries between Dot and Sum
Position
//                    pc.printf("\t\t%d\n\r",access.divBeforeMath);
//                    pc.printf("Dot detected before sub at: %d\n
\r",access.posDot_Before_Math);
                }
            }
            for (int dotAfterSub = access.posSubOperation + 1; dotAfterSub
<= k; dotAfterSub++)
            {
                if(access.calculationArray[dotAfterSub] == 66)
                {
                    access.posDot_After_Math = dotAfterSub;
                    access.secondDecimalPiont = true;
                    access.mulAfterMath = (access.posDot_After_Math -
access.posSubOperation - 1);
//                    pc.printf("\t\t%d\n\r",access.mulAfterMath);

                    access.divAfterMath = k - access.posDot_After_Math -1;
// number of entries between Dot and Sum Position
                  // pc.printf("\t\t%d\n\r",access.divAfterMath);
//                    pc.printf("Dot detected After sub at: %d\n
\r",access.posDot_After_Math);
//
                }
            }
        access.subFlag = false;
        }

        if (access.mulFlag == true)     // use the position on
posSumOperation() to find the position of the "." before the operator and
after the operator "*"
        {
            for(int dotBeforeMul = 0; dotBeforeMul <
access.posMulOperation; dotBeforeMul++)
            {
                if(access.calculationArray[dotBeforeMul] == 66)
                {
                    access.posDot_Before_Math = dotBeforeMul;
                    access.firstDecimalPiont = true;
                    access.mulBeforeMath = access.posDot_Before_Math;  //
number of entries between first entry and Dot
//                    pc.printf("\t\t%d\n\r",access.mulBeforeMath);

                    access.divBeforeMath = access.posMulOperation -
access.posDot_Before_Math - 1;  // number of entries between Dot and Sum
Position
//                    pc.printf("\t\t%d\n\r",access.divBeforeMath);
```

```
                    // pc.printf("Dot detected before mul at: %d\n
\r",access.posDot_Before_Math);
                }
            }
            for (int dotAfterMul = access.posMulOperation + 1; dotAfterMul
<= k; dotAfterMul++)
            {
                if(access.calculationArray[dotAfterMul] == 66)
                {
                    access.posDot_After_Math = dotAfterMul;
                    access.secondDecimalPiont = true;
                    access.mulAfterMath = (access.posDot_After_Math -
access.posMulOperation - 1);
//                    pc.printf("\t\t%d\n\r",access.mulAfterMath);

                    access.divAfterMath = k - access.posDot_After_Math -1;
// number of entries between Dot and Sum Position
//                    pc.printf("\t\t%d\n\r",access.divAfterMath);
//                    pc.printf("Dot detected After mul at: %d\n
\r",access.posDot_After_Math);

                }
            }
        access.mulFlag = false;
        }

        if (access.divFlag == true)     // use the position on
posSumOperation() to find the position of the "." before the operator and
after the operator "/"
        {
            for(int dotBeforeDiv = 0; dotBeforeDiv <
access.posDivOperation; dotBeforeDiv++)
            {
                if(access.calculationArray[dotBeforeDiv] == 66)
                {
                    access.posDot_Before_Math = dotBeforeDiv;
                    access.firstDecimalPiont = true;
                    access.mulBeforeMath = access.posDot_Before_Math;  //
number of entries between first entry and Dot
//                    pc.printf("\t\t%d\n\r",access.mulBeforeMath);

                    access.divBeforeMath = access.posDivOperation -
access.posDot_Before_Math - 1;  // number of entries between Dot and Sum
Position
                   // pc.printf("\t\t%d\n\r",access.divBeforeMath);
//                    pc.printf("Dot detected before div at: %d\n
\r",access.posDot_Before_Math);
                }
            }
            for (int dotAfterDiv = access.posDivOperation + 1; dotAfterDiv
<= k; dotAfterDiv++)
            {
                if(access.calculationArray[dotAfterDiv] == 66)
                {
```

```
                    access.posDot_After_Math = dotAfterDiv;
                    access.secondDecimalPiont = true;
                    access.mulAfterMath = (access.posDot_After_Math -
access.posDivOperation - 1);
//                    pc.printf("\t\t%d\n\r",access.mulAfterMath);

                    access.divAfterMath = k - access.posDot_After_Math -1;
// number of entries between Dot and Sum Position
//                    pc.printf("\t\t%d\n\r",access.divAfterMath);
//                    pc.printf("Dot detected After div at: %d\n
\r",access.posDot_After_Math);

                }
            }
        access.divFlag = false;
        }

        if (access.powFlag == true)     // use the position on
posSumOperation() to find the position of the "." before the operator and
after the operator "/"
        {
            for(int dotBeforePow = 0; dotBeforePow <
access.posPowOperation; dotBeforePow++)
            {
                if(access.calculationArray[dotBeforePow] == 66)
                {
                    access.posDot_Before_Math = dotBeforePow;
                    access.firstDecimalPiont = true;
                    access.mulBeforeMath = access.posDot_Before_Math;  //
number of entries between first entry and Dot
//                    pc.printf("\t\t%d\n\r",access.mulBeforeMath);

                    access.divBeforeMath = access.posPowOperation -
access.posDot_Before_Math - 1;  // number of entries between Dot and Sum
Position
//                    pc.printf("\t\t%d\n\r",access.divBeforeMath);
//                    pc.printf("Dot detected before pow at: %d\n
\r",access.posDot_Before_Math);
                }
            }
            for (int dotAfterPow = access.posPowOperation + 1; dotAfterPow
<= k; dotAfterPow++)
            {
                if(access.calculationArray[dotAfterPow] == 66)
                {
                    access.posDot_After_Math = dotAfterPow;
                    access.secondDecimalPiont = true;
                    access.mulAfterMath = (access.posDot_After_Math -
access.posPowOperation - 1);
//                    pc.printf("\t\t%d\n\r",access.mulAfterMath);

                    access.divAfterMath = k - access.posDot_After_Math -1;
// number of entries between Dot and Sum Position
//                    pc.printf("\t\t%d\n\r",access.divAfterMath);
```

```
//                    pc.printf("Dot detected After pow at: %d\n
\r",access.posDot_After_Math);

                }
            }
        access.powFlag = false;
        }

    return;
}


//*********************************************************************
**MAIN*************************************************************
***
int main()
{

    characterString *ptr = &access;
    clearFunction(ptr);  // Clear all the entries in Initial Character
Array


    pc.baud(921600);
    int j, t, k = 0;
    access.programCounter = 0;

    while(1)
    {

        pc.printf("\n\r");
        pc.printf("\n\r");

       for (access.programCounter = 0, j = 0; access.programCounter <
bufferZise; j++, access.programCounter++)
       {
            pc.scanf("%c", &access.array[access.programCounter]);

            if (access.array[access.programCounter] == 0x0D)  // If
'ENTER' is pressed we stop looking for entries
            {
//                clearFunction(ptr);
                access.programCounter =  bufferZise;
            }

            deleteFunction(ptr);

            if (access.array[access.programCounter] == 0x08)
            {
                access.array[access.programCounter] = 0;
                access.programCounter =  access.programCounter - 2;
                j = j - 2;

            }
```

```
            if (((access.array[access.programCounter] >=
0x01)&&(access.array[access.programCounter] < 0x08))||  // filter the
values we want
                ((access.array[access.programCounter] >=
0x09)&&(access.array[access.programCounter] < 0xD ))||  // to accept
                ((access.array[access.programCounter] >= 0xE)
&&(access.array[access.programCounter] < 0x2A))||  // ALL values here
                ((access.array[access.programCounter] >=
0x3A)&&(access.array[access.programCounter] < 0x5E))||  // are rejected
                ((access.array[access.programCounter] >=
0x5F)&&(access.array[access.programCounter]<= 0x7F))||
                ( access.array[access.programCounter] == 0x2C))
                {
                   lcd.cls();
                   pc.printf("ERROR: Entry NOT Valid!!!\n\r ");
                   lcd.printf("ERROR !!! ");
                   access.programCounter--;
                }

        }

        pc.printf("\n\r");  // "\t" is used because we need carrige return
on TeraTerm, Not only line feed
        pc.printf("\n\r");

        //*****************************************************Allocating
memory for two arrays*****************************************************
        access.calculationArray = (int*)calloc((j-1),
sizeof(int));//memory allocated using malloc if(ptr==NULL) {
printf("Error! memory not allocated."); exit(0); }
        if(access.calculationArray == NULL)
        {
            lcd.cls();
            lcd.printf("PRESS RESET BUTTON");
            pc.printf("ERROR!!! MEMORY NOT ALLOCATED!!! \t\n");
            return 0;
        }

        access.integerValuesArray = (int*)calloc((j-1),
sizeof(int));//memory allocated using malloc if(ptr==NULL) {
printf("Error! memory not allocated."); exit(0); }
        if(access.integerValuesArray == NULL)
        {
            lcd.cls();
            lcd.printf("PRESS RESET BUTTON");
            pc.printf("ERROR!!! MEMORY NOT ALLOCATED!!! \t\n");
            return 0;
        }
        //*****************************************************Allocating
memory for two arrays*****************************************************


       for (t = 0; t < j - 1; t++)
```

```
        {
             access.calculationArray[t] = access.array[t];

             if (access.array[0] == 0x0D)
             {
                 return main();
             }
             else
             {
//                 pc.printf("%d\t", access.array[t]);
//                 pc.printf("%c \t", access.array[t]);
//                 pc.printf("%f \n \r", array[t]);
//                 pc.printf("0x%02x \t", access.array[t]);
//                 val = access.array[i] * access.array[i];;
//                 pc.printf("% \n \r", val);

             }

        }

        //*****************************switching through decimal values to
fill integerValuesArray array*****************************
        for(int p = 0; p < j - 1; p++)
        {
             switch(access.calculationArray[p])
             {

                 case 42:
                     // 99 = "*" ==> miltiplication
                     access.mulFlag = true;
                     access.calculationArray[p] = 99;
                 break;

                 case 43:
                     // 88 = "+" ==> addition
                     access.sumFlag = true;
                     access.calculationArray[p] = 88;
                 break;

                 case 45:
                     // 77 = "-" ==> substranction
                     access.subFlag = true;
                     access.calculationArray[p] = 77;
                 break;

                 case 46:
                     // 66 = "." ==> decimal point
                     access.decPlaceFlag = true;
                     access.calculationArray[p] = 66;
                 break;

                 case 47:
                     // 55 = "/" ==> division
                     access.divFlag = true;
```

```
            access.calculationArray[p] = 55;
        break;

        case 48:
            access.calculationArray[p] = 0;
        break;

        case 49:
            access.calculationArray[p] = 1;
        break;

        case 50:
            access.calculationArray[p] = 2;
        break;

        case 51:
            access.calculationArray[p] = 3;
        break;

        case 52:
            access.calculationArray[p] = 4;
        break;

        case 53:
            access.calculationArray[p] = 5;
        break;

        case 54:
            access.calculationArray[p] = 6;
        break;

        case 55:
            access.calculationArray[p] = 7;
        break;

        case 56:
            access.calculationArray[p] = 8;
        break;

        case 57:
            access.calculationArray[p] = 9;
        break;

        case 94:
            // 44 = "^" ==> power
            access.powFlag = true;
            access.calculationArray[p] = 44;
        break;

        default:
            pc.printf("ERROR!!!\n\r");
        break;

    }
```

```
        }

        k = j - 1;

        //*****************************switching through decimal values to
fill integerValuesArray array*****************************




        //*********************** test if data gets saved onto the other
array************************
 //       for(int r = 0; r < j - 1; r++)
//        {
//            pc.printf("%d\n\r", access.calculationArray[r]);
//        }
        //*********************** test if data gets saved onto the other
array************************


        if (access.sumFlag == true)  // It has been detected that
        {
            decPlacedFunction(ptr, k);
            findNumbers(ptr,k);
            decimalPositionsFunction(ptr);
            sumFunction(ptr, k);

        }

        if (access.subFlag == true)
        {
            decPlacedFunction(ptr, k);
            findNumbers(ptr,k);
            subFunction(ptr, k);
            decimalPositionsFunction(ptr);
        }

        if (access.divFlag == true)
        {
            decPlacedFunction(ptr, k);
            findNumbers(ptr,k);
            divFunction(ptr, k);
            decimalPositionsFunction(ptr);
        }

        if (access.mulFlag == true)
        {
            decPlacedFunction(ptr, k);
            findNumbers(ptr,k);
            mulFunction(ptr, k);
            decimalPositionsFunction(ptr);
        }
```

```
        if (access.powFlag == true)
        {
            decPlacedFunction(ptr, k);
            findNumbers(ptr,k);
            powFunction(ptr, k);
            decimalPositionsFunction(ptr);
        }

        clearFunction(ptr);
        // ********************deallocating
memory************************************************
        free(access.calculationArray); // de-allocate the previously
allocated space
        free(access.integerValuesArray); // de-allocate the previously
allocated space
        // ********************deallocating
memory************************************************

        pc.printf("\r\n TYPE, THEN PRESS <ENTER> TWICE:\r\n");
    }
return 0;
}
```

Attached Files:

The following files have been attached in order for the program to compile and run:

main.cpp

TextLCD.cpp

TextLCD.h

NOTE:

As the source code has been copied onto word, the comments and lines that were wider than the word page, have automatically jumped to the next line. This does not mean that the program is written exactly like it looks in word. Look at the source code file (main.cpp) for the appropriate format of the code.

Results:

As expected the calculator worked as anticipated. Out calculator was capable of performing the mathematical calculations with no problem, as specified by the description of this project. During performing the *Power() or "^"* calculation, we found that the library used by the online compiler, "mbed.h", did not support the built-in **pow(),** when a number was raised to a very small power, e.g. *pow(2, 0.0001).* This caused problems at the beginning, but they were solved; a different approach was taken.

When debugging the program, we encountered problems regarding memory allocation and, as needed, is was fixed; we made sure that the allocated memory was being de-allocated after the calculation.

It can be said that the project was successfully implemented. We added capabilities to the programs as specified above, but note that more capabilities such as finding the $Log_n()$, or even binary calculations capabilities.