

Calculator

Pothuri Rahul - EE24BTECH11050

1 Introduction

This project presents a calculator using Arduino , embedded C language . This calculator is designed so that we can compute many mathematical operations like arithmetic operations , trigonometric functions , inverse trigonometric functions , power function , etc.This should be done without using math.h library , hence used some different methods to compute few functions.

2 Hardware things

2.1 Components used :

- Breadboard
- LCD display
- Arduino board
- Connecting cables and wires
- Mobile
- push buttons

2.2 Circuit design

2.2.1 LCD to Arduino

Following connections are done between LCD and Arduino

LCD display	Arduino
RS	8
Enable(E)	9
D_4	10
D_5	11
D_6	12
D_7	13
RW	GND
VSS	GND
VDD	5 V
VEE	Potentiometer (Middle Pin)

2.2.2 Push buttons to Arduino

Let us consider each 6 buttons as a row and their are 6 such rows (A total of 36 buttons) . Each row and column are connected as follows .

ROW	Arduino	Column	Arduino
Row 1	2	Column 1	A_0
Row 2	3	Column 2	A_1
Row 3	4	Column 3	A_2
Row 4	5	Column 4	A_3
Row 5	6	Column 5	A_4
Row 6	7	Column 6	A_5

Functionality of each push button is as follows.

Row	Col 1 (A0)	Col 2 (A1)	Col 3 (A2)	Col 4 (A3)	Col 5 (A4)	Col 6 (A5)
R1 (2)	0	6	*	sin	e	y^x
R2 (3)	1	7	/	cos	log	z
R3 (4)	2	8	=	tan	ln	π
R4 (5)	3	9	C (Clear)	cot	sqrt	natural log
R5 (6)	4	+	i	sec	cosec	Answer
R6 (7)	5	-	.	\sin^{-}	\cos^{-}	\tan^{-}

Table 1: 6×6 Keypad Button Mapping

3 Software things

3.1 Code used

The c code used is the following

```

1  #define F_CPU 16000000UL
2  #include <avr/io.h>
3  #include <util/delay.h>
4  #include <stdlib.h>
5  #include <stdbool.h>
6  #include <stdio.h>
7  #include <string.h>
8
9  // TYPEDEFS
10 typedef uint8_t byte;
11
12 // LCD PINS
13 #define ClearBit(x,y) x &= ~_BV(y)
14 #define SetBit(x,y) x |= _BV(y)
15 #define LCD_RS 0
16 #define LCD_E 1
17 #define DAT4 2
18 #define DAT5 3
19 #define DAT6 4
20 #define DAT7 5
21 #define CLEARDISPLAY 0x01
22
23 #define ROWS 6
24 #define COLS 6
25
26 #define K 0.607252935
27 #define NUM_STEPS 15
28
29 // Define the macro before using it
30 #define ANGLE_OUT_DEGREES { \
31     45.00000000000000, /* atan(2^(-0)) */ \
32     26.56505117707799, /* atan(2^(-1)) */ \
33     14.03624346792648, /* atan(2^(-2)) */ \
34     7.125016348901798, /* atan(2^(-3)) */ \
35     3.576334374997351, /* atan(2^(-4)) */ \
36     1.789910608246069, /* atan(2^(-5)) */ \
37     0.8951737102110744, /* atan(2^(-6)) */ \
38     0.4476141708605531, /* atan(2^(-7)) */ \
39     0.2238105003685381, /* atan(2^(-8)) */ \
40     0.1119056770662069, /* atan(2^(-9)) */ \
41     0.05595289189380368, /* atan(2^(-10)) */ \
42     0.02797645261700364, /* atan(2^(-11)) */ \
43     0.013988227142265016, /* atan(2^(-12)) */ \
44     0.006994113675352919, /* atan(2^(-13)) */ \
45     0.003497056850704011 /* atan(2^(-14)) */ \
46 }
47
48 // Then use it to initialize the array
49 double angle_out_degrees[] = ANGLE_OUT_DEGREES;
50
51 void PulseEnableLine(void) {
52     SetBit(PORTB, LCD_E);
53     _delay_us(40);
54     ClearBit(PORTB, LCD_E);
55 }
56
57 void SendNibble(byte data) {
58     PORTB &= 0xC3; // Clear 4 data lines

```

```

59     if (data & _BV(4)) SetBit(PORTB, DAT4);
60     if (data & _BV(5)) SetBit(PORTB, DAT5);
61     if (data & _BV(6)) SetBit(PORTB, DAT6);
62     if (data & _BV(7)) SetBit(PORTB, DAT7);
63     PulseEnableLine();
64 }
65
66 void SendByte(byte data) {
67     SendNibble(data);
68     SendNibble(data << 4);
69 }
70
71 void LCD_Cmd(byte cmd) {
72     ClearBit(PORTB, LCD_RS);
73     SendByte(cmd);
74 }
75
76 void LCD_Char(byte ch) {
77     SetBit(PORTB, LCD_RS);
78     SendByte(ch);
79 }
80
81 void LCD_Init() {
82     LCD_Cmd(0x33);
83     LCD_Cmd(0x32);
84     LCD_Cmd(0x28);
85     LCD_Cmd(0x0C);
86     LCD_Cmd(0x06);
87     LCD_Cmd(0x01);
88     _delay_ms(3);
89 }
90
91 void LCD_Clear() {
92     LCD_Cmd(CLEARDISPLAY);
93     _delay_ms(3);
94 }
95
96 void LCD_Message(const char *text) {
97     while (*text) LCD_Char(*text++);
98 }
99
100 void LCD_Float(double data) {
101     char st[16];
102     dtostrf(data, 6, 2, st);
103     LCD_Message(st);
104 }
105
106 // VARIABLES
107 volatile double Num1 = 0.0, Num2 = 0.0, Number = 0.0;
108 volatile double answer = 0.0; // Added variable to store the last result
109 volatile char action = 0;
110 volatile uint8_t result = 0;
111 volatile uint8_t inputMode = 0;
112 volatile bool opt1 = 0, opt2 = 0, decimal_mode = false, shift = false;
113 volatile double decimal_factor = 0.1;
114 volatile uint8_t return_act = 0;
115 volatile bool math_error = false; // New flag to track math errors
116 double x = 0;
117 double y = 0;

```

```

118 double angle = 0.0; // Added missing global variable declaration
119
120 // BUTTONS MAPPING
121 uint8_t row_pins[ROWS] = {PD0, PD1, PD2, PD3, PD4, PD5};
122 uint8_t col_pins[COLS] = {PC0, PC1, PC2, PC3, PC4, PC5};
123
124 char keypad[ROWS][COLS] = {
125     {'0', '6', '*', 's', 'e', 'y'},
126     {'1', '7', '/', 'c', 'l', 'z'},
127     {'2', '8', '=', 't', 'a', 'p'},
128     {'3', '9', 'C', 'q', 'b', 'L'},
129     {'4', '+', 'i', 'm', 'w', 'A'},
130     {'5', '-', '.', 'n', 'x', 'S'}
131 };
132
133 void keypad_init() {
134     // Configure row pins as input with pull-up resistors (Now using PORTD)
135     for (int i = 0; i < ROWS; i++) {
136         DDRD &= ~(1 << row_pins[i]); // Set as input
137         PORTD |= (1 << row_pins[i]); // Enable pull-up
138     }
139
140     // Configure column pins as output
141     for (int j = 0; j < COLS; j++) {
142         DDRC |= (1 << col_pins[j]); // Set as output
143         PORTC |= (1 << col_pins[j]); // Set high
144     }
145 }
146
147 uint8_t keypad_scan() {
148     for (int col = 0; col < COLS; col++) {
149         // Set all columns high
150         for (int j = 0; j < COLS; j++) {
151             PORTC |= (1 << col_pins[j]);
152         }
153
154         // Set current column low
155         PORTC &= ~(1 << col_pins[col]);
156
157         for (int row = 0; row < ROWS; row++) {
158             if (!(PIND & (1 << row_pins[row]))) { // Key is pressed
159                 _delay_ms(50); // Debounce
160                 while (!(PIND & (1 << row_pins[row]))); // Wait for release
161                 return keypad[row][col]; // Return key value
162             }
163         }
164     }
165     return 0; // No key pressed
166 }
167
168 double ln(double x) {
169     if (x <= 0) {
170         math_error = true; // Set math error flag
171         return 0; // Logarithm undefined for non-positive numbers
172     }
173
174     double y = x - 1.0; // Initial approximation
175     for (int i = 0; i < 1000; i++) { // More iterations improve accuracy
176         y = y - (exp(y) - x) / exp(y) - 1;

```

```

177     }
178     return y;
179 }
180
181 double exp(double x) {
182     double y = 1.0; // Initial condition:  $e^0 = 1$ 
183     double h = 0.01; // Step size for approximation
184     int steps = (int)(x / h); // Number of steps
185
186     if (x < 0) { // Handle negative exponent using inverse
187         x = -x;
188         for (int i = 0; i < steps; i++) {
189             y *= (1 + h);
190         }
191         return 1.0 / y;
192     }
193
194     for (int i = 0; i < steps; i++) {
195         y *= (1 + h);
196     }
197
198     return y;
199 }
200
201 // Custom square root function since we can't use math.h
202 double my_sqrt(double x) {
203     if (x < 0) {
204         math_error = true;
205         return 0;
206     }
207
208     if (x == 0) return 0;
209
210     // Newton's method for square root
211     double guess = x / 2.0;
212     double prev_guess;
213
214     for (int i = 0; i < 10; i++) {
215         prev_guess = guess;
216         guess = (guess + x / guess) / 2.0;
217
218         // Check for convergence
219         if (fabs(guess - prev_guess) < 0.0001)
220             break;
221     }
222
223     return guess;
224 }
225
226 // Improved power function
227 double power(double base, double exponent) {
228     // Special case: anything0 = 1 (except 00 which is undefined)
229     if (exponent == 0) {
230         if (base == 0) {
231             math_error = true; // 00 is undefined
232             return 0;
233         }
234         return 1.0;
235     }

```

```

236 // Special case: 0^anything = 0 (except 0^negative which is undefined)
237
238 if (base == 0) {
239     if (exponent < 0) {
240         math_error = true; // Division by zero
241         return 0;
242     }
243     return 0;
244 }
245
246 // Handle integer exponents directly for better accuracy
247 bool is_int_exponent = (fabs(exponent - round(exponent)) < 0.000001);
248 double int_exp = round(exponent);
249
250 if (is_int_exponent && int_exp > 0 && int_exp <= 10) {
251     // Direct calculation for small positive integer exponents
252     double result = 1.0;
253     for (int i = 0; i < int_exp; i++) {
254         result *= base;
255     }
256     return result;
257 }
258
259 // Handle negative base with integer exponent
260 if (base < 0) {
261     if (!is_int_exponent) {
262         math_error = true; // Complex result for non-integer exponent
263         return 0;
264     }
265
266     // For negative base with integer exponent
267     double result = exp(exponent * ln(-base));
268     return (fmod(int_exp, 2) == 0) ? result : -result;
269 }
270
271 // Use logarithm method for other cases
272 return exp(exponent * ln(base));
273 }
274
275 void sin_cos(double n) {
276     // Declare necessary variables
277     x = 1.0;
278     y = 0.0;
279     double angle = 0.0;
280
281     // CORDIC algorithm for sin and cos calculation
282     for (uint8_t i = 0; i < NUM_STEPS; i++) {
283         int sigma = (angle < n) ? 1 : -1;
284
285         double scale = 1.0 / (1UL << i); // Precompute scale = 2^-i
286         double x_new = x - sigma * (y * scale);
287         double y_new = y + sigma * (x * scale);
288
289         x = x_new;
290         y = y_new;
291
292         angle += sigma * angle_out_degrees[i];
293     }
294 }

```

```

295     x = x * K;
296     y = y * K;
297 }
298
299 void inv_trigo(double z, char mode) {
300     // Declare necessary variables
301     x = 1.0;
302     y = 0.0;
303     angle = 0.0;
304
305     // Input validation: z must be between -1 and 1 for arcsin/arccos
306     if ((mode == 'a' || mode == 'b') && (z < -1.0 || z > 1.0)) {
307         math_error = true; // Set math error flag
308         return;
309     }
310
311     // Initialize starting point based on the inverse function type
312     switch (mode) {
313         case 'a': // arcsin
314             x = my_sqrt(1-z*z);
315             y = z;
316             break;
317         case 'b': // arccos
318             x = z;
319             y = my_sqrt(1-z*z);
320             break;
321         case 'w': // arctan
322             x = 1;
323             y = z;
324             break;
325         case 'x': // arccot
326             x = z;
327             y = 1;
328             break;
329         case 'y': // arccsc
330             if (fabs(z) < 1.0) {
331                 math_error = true; // Set math error flag
332                 return; // Invalid domain
333             }
334             x = my_sqrt(z*z-1)/fabs(z);
335             y = 1/fabs(z);
336             if (z < 0) y = -y;
337             break;
338         case 'z': // arcsec
339             if (fabs(z) < 1.0) {
340                 math_error = true; // Set math error flag
341                 return; // Invalid domain
342             }
343             x = 1/fabs(z);
344             y = my_sqrt(z*z-1)/fabs(z);
345             if (z < 0) x = -x;
346             break;
347         // Handle unexpected values
348     }
349
350     // Now rotate back to the x-axis
351     for (uint8_t i = 0; i < NUM_STEPS; i++) {
352         // Determine rotation direction to reduce y toward 0
353         int sigma = (y < 0) ? 1 : -1;

```



```

354
355         double scale = 1.0 / (1UL << i); // Precompute scale = 2-i
356         double x_new = x - sigma * (y * scale);
357         double y_new = y + sigma * (x * scale);
358
359         x = x_new;
360         y = y_new;
361
362         // Accumulate rotation angle
363         angle += sigma * angle_out_degrees[i];
364     }
365 }
366
367 void calculate_result() {
368     math_error = false; // Reset error flag at start of calculation
369
370     if (action == '+') Number = Num1 + Num2;
371     else if (action == '-') Number = Num1 - Num2;
372     else if (action == '*') Number = Num1 * Num2;
373     else if (action == 'p') Number = power(Num1, Num2);
374     else if (action == 'h') {
375         if (Num2 == 0) {
376             math_error = true;
377             Number = 0;
378         } else if (Num2 == 2) {
379             Number = my_sqrt(Num1); // Square root (2nd root)
380         } else {
381             // For nth root, use: x(1/n)
382             Number = power(Num1, 1.0/Num2);
383         }
384     }
385     else if (action == 'L') {
386         if (Num1 <= 0 || Num2 <= 0 || fabs(Num1 - 1.0) < 0.000001) {
387             math_error = true; // Set math error flag for invalid inputs
388             Number = 0;
389         } else {
390             Number = ln(Num2)/ln(Num1);
391         }
392     }
393     else if (action == '/') {
394         if (Num2 == 0) {
395             math_error = true; // Set math error flag
396             Number = 0;
397         } else {
398             Number = Num1 / Num2;
399         }
400     }
401     else if (action == 's' || action == 'c' || action == 't' || action == 'q' || action == 'm' || action == 'n')
402         sin_cos(Num2);
403     switch(action) {
404         case 's': Number = y; // sine
405             break;
406         case 'c': Number = x; // cosine
407             break;
408         case 't':
409             if (fabs(x) < 0.000001) { // Check for division by zero (cos 0)
410                 math_error = true;
411                 Number = 0;
412             } else {

```

```

413         Number = y/x; // tangent
414     }
415     break;
416     case 'q':
417         if (fabs(y) < 0.000001) { // Check for division by zero (sin 0)
418             math_error = true;
419             Number = 0;
420         } else {
421             Number = x/y; // cotangent
422         }
423     break;
424     case 'm':
425         if (fabs(y) < 0.000001) { // Check for division by zero (sin 0)
426             math_error = true;
427             Number = 0;
428         } else {
429             Number = 1/y; // cosecant
430         }
431     break;
432     case 'n':
433         if (fabs(x) < 0.000001) { // Check for division by zero (cos 0)
434             math_error = true;
435             Number = 0;
436         } else {
437             Number = 1/x; // secant
438         }
439     break;
440 }
441 }
442 else if (action == 'a' || action == 'b' || action == 'w' || action == 'x' || action == 'y' || action == 'z')
443     inv_trigo(Num2, action);
444     Number = -angle;
445 }
446 else if (action == 'r' ) {
447     Number = (exp(Num2) - exp(-Num2))/2;
448 }
449 else if (action == 'g' ) {
450     Number = (exp(Num2) + exp(-Num2))/2;
451 }
452 else if(action == 'e'){
453     Number = exp(Num2);
454 }
455 else if(action == 'l'){
456     Number = ln(Num2);
457 }
458
459 // Store the result in the answer variable if no math error occurred
460 if (!math_error) {
461     answer = Number;
462 }
463
464 result = 1;
465 }
466
467 void process_input(char key) {
468     if (key == 'i') {
469         shift = !shift;
470     }
471     else if (key == 'S') {

```

```

472         // Toggle the sign of the current number
473         Number = -Number;
474     }
475     else if (key == 'A') {
476         // Recall the stored answer value
477         Number = answer;
478         LCD_Clear();
479         LCD_Message("Ans: ");
480         LCD_Float(answer);
481         _delay_ms(1000);
482     }
483     else if (key == 'C') {
484         if(shift == 0){
485             Number = Num1 = Num2 = 0;
486             action = 0;
487             result = 0;
488             decimal_mode = false;
489             decimal_factor = 0.1;
490             shift = 0;
491             math_error = false; // Reset error flag
492             // Note: We don't reset the answer variable to preserve it across calculations
493             LCD_Clear();
494             LCD_Message("Cleared");
495             _delay_ms(1000);
496             LCD_Clear();
497             return;
498         }
499         else{
500             if (!decimal_mode) {
501                 // Remove last digit in integer mode
502                 Number = (Number < 0) ? -((int)(-Number / 10)) : (int)(Number / 10);
503             } else {
504                 // Separate integer and decimal parts
505                 double intPart, fracPart;
506                 fracPart = modf(Number, &intPart);
507
508                 // Track decimal precision
509                 static int decimal_places = 0;
510
511                 if (fracPart != 0) {
512                     // If we have decimal digits, remove the last one
513                     decimal_places--;
514
515                     if (decimal_places <= 0) {
516                         // If no more decimal places, switch back to integer mode
517                         Number = intPart;
518                         decimal_mode = false;
519                         decimal_places = 0;
520                     } else {
521                         // Round to the remaining decimal places
522                         double multiplier = power(10, decimal_places);
523                         fracPart = round(fracPart * multiplier) / multiplier;
524                         Number = (Number < 0) ? -(abs(intPart) + abs(fracPart)) : (intPart + fracPart);
525                     }
526                 } else {
527                     // No decimal part, switch back to integer mode
528                     decimal_mode = false;
529                     decimal_places = 0;
530                 }

```

```

531     }
532     }
533 }
534 else if (key >= '0' && key <= '9') {
535     if (result) { // If a calculation was done, start fresh
536         Number = 0;
537         result = 0;
538         decimal_mode = false;
539         decimal_factor = 0.1;
540         math_error = false; // Reset error flag
541     }
542     if (!decimal_mode) {
543         Number = (Number * 10) + (key - '0');
544         // Preserve the sign when adding digits
545         if (Number < 0) {
546             Number = -fabs(Number);
547         }
548     } else {
549         // Add decimal digits while preserving sign
550         if (Number < 0) {
551             Number -= (key - '0') * decimal_factor;
552         } else {
553             Number += (key - '0') * decimal_factor;
554         }
555         decimal_factor *= 0.1;
556     }
557 }
558 else if (key == '.') {
559     if (!decimal_mode) {
560         decimal_mode = true;
561     }
562 }
563
564 else if (key == '+' || key == '-' || key == '*' || key == '/' || key == 'L') {
565     Num1 = Number;
566     Number = 0;
567     action = key;
568     decimal_mode = false;
569     decimal_factor = 0.1;
570     math_error = false; // Reset error flag
571 }
572
573 else if (key == 'p'){
574     if(shift == 0){
575         action = 'p';
576         Num1 = Number;
577         Number = 0;
578         decimal_mode = false;
579         decimal_factor = 0.1;
580         math_error = false;
581     }
582     else{
583         action = 'h'; // Root operation
584         Num1 = Number; // Store the number we want to find the root of
585         Number = 0; // Reset number for entering the root order
586         decimal_mode = false;
587         decimal_factor = 0.1;
588         math_error = false;
589         // Don't reset shift here - let the display show we're in root mode

```

```

590     }
591 }
592
593 else if (key == 'p'){
594     if(shift == 0){
595         action = 'p';
596         Num1 = Number;
597         Number = 0;
598         decimal_mode = false;
599         decimal_factor = 0.1;
600         math_error = false;
601     }
602     else{
603         action = 'h'; // Root operation
604         Num1 = Number; // Store the number we want to find the root of
605         Number = 0;    // Reset number for entering the root order
606         decimal_mode = false;
607         decimal_factor = 0.1;
608         math_error = false;
609         // Don't reset shift here - let the display show we're in root mode
610     }
611 }
612
613 else if (key == 's' || key == 'c' || key == 't' || key == 'q' || key == 'm' || key == 'n') {
614     if(shift == 0){
615         switch(key) {
616             case 's': action = 's';
617                     break;
618             case 'c': action = 'c';
619                     break;
620             case 't': action = 't';
621                     break;
622             case 'q': action = 'q';
623                     break;
624             case 'm': action = 'm';
625                     break;
626             case 'n': action = 'n';
627                     break;
628         }
629     }
630     else{
631         switch(key) {
632             case 's': action = 'r';
633                     break;
634             case 'c': action = 'g';
635                     break;
636         }
637     }
638 }
639 else if(key == 'a' || key == 'b' || key == 'w' || key == 'x' || key == 'y' || key == 'z'){
640     switch(key) {
641         case 'a': action = 'a';
642                 break;
643         case 'b': action = 'b';
644                 break;
645         case 'w': action = 'w';
646                 break;
647         case 'x': action = 'x';
648                 break;

```

```

649         case 'y': action = 'y';
650             break;
651         case 'z': action = 'z';
652             break;
653     }
654
655     Number = 0; // Reset Number to avoid displaying 0.00
656     decimal_mode = false;
657     decimal_factor = 0.1;
658     math_error = false; // Reset error flag
659 }
660 else if(key == 'e'){
661     if(shift == 0){
662         action = 'e';
663         Number = 0; // Reset Number to avoid displaying 0.00
664         decimal_mode = false;
665         decimal_factor = 0.1;
666         math_error = false; // Reset error flag
667     }
668     else {
669         Number = exp(1); // Set Number to e
670         shift = 0; // Reset shift flag after using it
671
672         // Special flag to indicate we should display 'e' instead of the value
673         action = 'E'; // Using capital 'E' to indicate special display mode
674     }
675 }
676 else if(key == 'l'){
677     if(shift == 0){
678         action = 'l';
679         Number = 0; // Reset Number to avoid displaying 0.00
680         decimal_mode = false;
681         decimal_factor = 0.1;
682         math_error = false; // Reset error flag
683     }
684     else {
685         Number = 3.14159265358979; // Set Number to pi
686         shift = 0; // Reset shift flag after using it
687
688         // Special flag to indicate we should display '' instead of the value
689         action = 'P'; // Using capital 'L' to indicate special display mode for pi
690     }
691 }
692 else if (key == '=') {
693     Num2 = Number;
694     calculate_result();
695     decimal_mode = false;
696     decimal_factor = 0.1;
697 }
698 }
699
700 int main(void) {
701     keypad_init();
702     DDRB |= (1 << LCD_RS) | (1 << LCD_E) | (1 << DAT4) | (1 << DAT5) | (1 << DAT6) | (1 << DAT7);
703     LCD_Init();
704     LCD_Clear();
705     LCD_Float(Number);
706
707     while (1) {

```

```

708     char key = keypad_scan();
709     if (key) {
710         process_input(key);
711         LCD_Clear();
712
713         if (result == 1) {
714             result = 0;
715
716             if (math_error) {
717                 // Display "Math Error" if a math error occurred
718                 LCD_Message("Math Error");
719                 _delay_ms(1500); // Display error message a bit longer
720
721                 // Reset all calculation values except answer
722                 Number = Num1 = Num2 = 0;
723                 action = 0;
724                 decimal_mode = false;
725                 decimal_factor = 0.1;
726                 math_error = false;
727
728                 LCD_Clear();
729                 LCD_Float(0); // Display zero after error
730             } else {
731                 // No error, display result normally
732                 LCD_Message("Result:");
733                 _delay_ms(100);
734                 LCD_Float(Number);
735                 _delay_ms(100);
736             }
737         }
738         else if (action) {
739             // Fixed display logic with proper structure
740             if (action == 'E') {
741                 LCD_Message("e");
742             }
743             else if (action == 'h') { // Root display
744                 if (Num2 != 0) {
745                     // When Num2 is set, display the complete expression
746                     LCD_Float(Num2);
747                     LCD_Message("-root(");
748                     LCD_Float(Num1);
749                     LCD_Message(")");
750                 } else {
751                     // When just starting the root operation
752                     LCD_Message("root(");
753                     LCD_Float(Num1);
754                     LCD_Message(",");
755                     if (Number != 0) {
756                         LCD_Float(Number);
757                     }
758                 }
759             }
760             else if (action == 'p') {
761                 LCD_Float(Num1);
762                 LCD_Message("^");
763                 if (Number != 0) {
764                     LCD_Float(Number);
765                 }
766             }

```

```

767     else if (action == 'L' && shift == 0) { // Regular logarithm
768         LCD_Message("log_");
769         LCD_Float(Num1);
770         LCD_Message("(");
771         if (Number != 0) {
772             LCD_Float(Number);
773             LCD_Message(")");
774         }
775     }
776     else if (action == 'P') { // Pi constant
777         LCD_Message("pi");
778     }
779     // First check for inverse trig functions (they need special handling)
780     else if (action == 'e') {
781         LCD_Message("exp(");
782         LCD_Float(Number);
783         LCD_Message(")");
784     }
785     else if (action == 'l') {
786         LCD_Message("ln(");
787         LCD_Float(Number);
788         LCD_Message(")");
789     }
790     else if (action == 'a' || action == 'b' || action == 'w' || action == 'x' || action == 'y' || action == 'z') {
791         switch(action) {
792             case 'a': LCD_Message("asin("); break;
793             case 'b': LCD_Message("acos("); break;
794             case 'w': LCD_Message("atan("); break;
795             case 'x': LCD_Message("acot("); break;
796             case 'y': LCD_Message("acosec("); break;
797             case 'z': LCD_Message("asec("); break;
798         }
799         if (Number != 0) {
800             LCD_Float(Number);
801             LCD_Message(")");
802         }
803     }
804     // Then check for standard trig functions
805     else if (action == 's' || action == 'c' || action == 't' || action == 'q' || action == 'm' || action == 'n' || action == 'r' || action == 'g') {
806         switch(action) {
807             case 's': LCD_Message("sin("); break;
808             case 'c': LCD_Message("cos("); break;
809             case 't': LCD_Message("tan("); break;
810             case 'q': LCD_Message("cot("); break;
811             case 'm': LCD_Message("cosec("); break;
812             case 'n': LCD_Message("sec("); break;
813             case 'r': LCD_Message("sinh("); break;
814             case 'g': LCD_Message("cosh("); break;
815         }
816         if (Number != 0) {
817             LCD_Float(Number);
818             LCD_Message(")");
819         }
820     }
821     // Finally handle basic operations
822     else {
823         LCD_Float(Num1);
824         _delay_ms(100);
825         LCD_Char(action);

```



```

826         _delay_ms(100);
827         if (Number != 0) {
828             LCD_Float(Number);
829         }
830     }
831 } else {
832     LCD_Float(Number);
833     _delay_ms(100);
834 }
835 }
836 }
837 }
838

```

Reference : Used code done by Yamasani Harsha Vardhan Reddy - ee24btech11063.

3.2 Explanation of the code

The given Arduino code implements a digital clock (HH:MM:SS) using a 7447 BCD to 7-segment decoder with multiplexing.

3.2.1 Key Features

1. LCD Handling (4-bit Mode)

- Initializes and controls an **LCD display** using **PORTB**.
- Functions like `LCD_Init()`, `LCD_Clear()`, and `LCD_Message()` handle output.

2. Keypad Scanning (6×6 Matrix)

- `keypad_init()`: Configures **6 row inputs (PORTD)** and **6 column outputs (PORTC)**.
- `keypad_scan()`: Detects pressed buttons by scanning rows and columns.

3. Mathematical Operations

- Implements basic (+, -, *, /) and advanced operations (log, exp, sqrt).
- **Trigonometry** (sin, cos, tan) and **inverse functions** (asin, acos) are computed using the **CORDIC algorithm**.

4. Handling User Input

- `process_input(char key)`: Determines the action based on the pressed button.
- Supports **Shift mode** for accessing extra functions (\sin^{-1} , e^x , π).
- Pressing = triggers `calculate_result()`, which processes the stored operation.

5. Result Calculation & Display

- `calculate_result()`: Executes the selected mathematical function.
- Displays results on **LCD**, with **error handling** for invalid inputs (e.g., division by zero).

6. Memory & Error Handling

- Stores the **previous answer** (ANS).
- Handles **math errors** like $\log(0)$ or division by zero.

3.2.2 How It Works

1. **User presses buttons** → Detected via `keypad_scan()`.
2. **Input is processed** → Determines the type of operation.
3. **Calculation is performed** → `calculate_result()` executes math operations.
4. **Result is displayed** → LCD shows the computed value.
5. **Shift Mode** allows access to extra functions (e.g., \sin^{-1} , \log).

4 Future Enhancements

- Adding differentiation and integration operations .
- Finding zeros of given expression or solution of a given equation .
- Adding charging battery system to make the calculator portable.
- Adding degree or radian mode i.e, by using a toggle button between them .

5 Conclusion

By doing this project I learned how to use c language to built mathematical functions without using `math.h` . This project successfully implemented a fully functional calculator using an Arduino Uno, push buttons , and LCD display.

Acknowledgment

I thank Yamasani Harsha Vardhan Reddy - ee24btech11063 for helping me in code.