Implementation of Scientific Calculator Functions on Microcontroller using C

Krishna Patil, Nara Prajwal
Department of Electrical Engineering
IIT Hyderabad
Email: ee24btech11036@iith.ac.in, ee24btech11051@iith.ac.in

***Abstract*—This paper presents the implementation of scientific calculator functions such as trigonometric, logarithmic, exponential, power, factorial, and expression parsing using the C programming language on a microcontroller platform. Various numerical algorithms such as Runge–Kutta (RK4), series expansions, iterative methods, and parsing algorithms have been employed. Each function is discussed with its algorithmic basis followed by its C implementation.**

***Index Terms*—Scientific Calculator, Microcontroller, Runge–Kutta Method, Quake III Algorithm, Shunting Yard Algorithm, C Programming**

I. Software Implementation

Instead of using "general expansion methods" (like Taylor/Maclaurin series with many terms), we reformulated each function as a differential equation. The algorithms used to calculate the values are RK4 (for ODEs), Newton–Raphson (for refinements like inverse sqrt).

A. Supported Functions

TABLE I
Supported Functions and Operations

Category	Functions / Operations
Trigonometric	$\sin(x), \cos(x), \tan(x)$
Inverse Trigonometric	$\arcsin(x), \arccos(x), \arctan(x)$
Exponential / Logarithmic	$e^x, \ln(x)$
Power / Root	x^y, $\frac{1}{\sqrt{x}}$
Factorial	$n!$
Basic Arithmetic	$+, -, \times, \div$
Constants	π, e
Input Symbols	Digits 0–9, decimal point, parentheses $(,)$

B. Runge–Kutta Method (RK4)

We employ the fourth-order Runge–Kutta (RK4) method [1], [3] to solve ordinary differential equations for trigonometric and exponential functions. The full derivation and step-by-step procedure are provided in Appendix A.

C. Trigonometric Functions

1) Sine Function: The sine function [4] is computed by solving the second-order differential equation:

$$\frac{d^2y}{dx^2} + y = 0, \quad y(0) = 0, \quad y'(0) = 1 \tag{1}$$

using the fourth-order Runge–Kutta method.

System:

$$\frac{dy}{dx} = z, \quad \frac{dz}{dx} = -y, \quad y(0) = 0,\ z(0) = 1 \tag{2}$$

The k_1, k_2, k_3, k_4 corresponds to the RK-4 terms of the equation $\frac{dy}{dx} = z$ and the l_1, l_2, l_3, l_4 corresponds to the RK-4 terms of the equation $\frac{dz}{dx} = -y$

RK4 step values:

$$k_1 = hz_n, \tag{3}$$
$$l_1 = -hy_n, \tag{4}$$
$$k_2 = h\left(z_n + \tfrac{l_1}{2}\right), \tag{5}$$
$$l_2 = -h\left(y_n + \tfrac{k_1}{2}\right), \tag{6}$$
$$k_3 = h\left(z_n + \tfrac{l_2}{2}\right), \tag{7}$$
$$l_3 = -h\left(y_n + \tfrac{k_2}{2}\right), \tag{8}$$
$$k_4 = h(z_n + l_3), \tag{9}$$
$$l_4 = -h(y_n + k_3), \tag{10}$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{11}$$
$$z_{n+1} = z_n + \tfrac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4). \tag{12}$$

2) Cosine Function: The cosine function [4] is computed by $\sin(\frac{\pi}{2} - x)$.

3) Tangent Function: Tangent [4] is implemented as the ratio of sine and cosine:

D. Inverse Square Root

1) Inverse Square Root: The fast inverse square root algorithm (Quake III method) is used to efficiently compute $x^{-1/2}$. Detailed derivation and Newton–Raphson refinement are included in Appendix A.

E. Inverse Trigonometric Functions

1) Arcsine: The ODE for the arcsine function [4] is ODE:

$$\frac{dy}{dx} = \frac{1}{\sqrt{1 - x^2}}, \quad y(0) = 0 \tag{13}$$

2) Arccosine: Use the identity [4]

$$\arccos(x) = \frac{\pi}{2} - \arcsin(x) \tag{14}$$

3) Arctangent: The ODE for the arctan function [4] is ODE:

$$\frac{dy}{dx} = \frac{1}{1 + x^2}, \quad y(0) = 0 \tag{15}$$

F. Logarithmic Functions

The natural logarithm can be obtained by solving the differential equation [4]

$$\frac{dy}{dx} = \frac{1}{x}, \quad y(1) = 0 \tag{16}$$

which has the exact solution $y(x) = \ln(x)$. Using the RK4 method, we can approximate $\ln(x)$ by integrating this ODE from $x = 1$ to the target value.

G. Exponential Functions

1) Power Function: The power function $y = x^w$ [4] satisfies

$$\frac{dy}{dx} = \frac{w}{x}y, \quad y(1) = 1 \tag{17}$$

H. Factorial Function

The algorithm is a simple recursive function in C

$$y_n = n \cdot y_{n-1} \tag{18}$$

[2]

Algorithm 1 Recursive Factorial Function

```
function FACTORIALRECURSIVE(x)
    if x ≤ 1 then
        return 1
    else
        return x × FactorialRecursive(x − 1)
    end if
end function
```

I. Expression Parsing

The Shunting Yard algorithm is employed for converting infix expressions into postfix form for evaluation.

J. Overview

The **Shunting Yard Algorithm**, proposed by Edsger Dijkstra, is a method for converting mathematical expressions in infix notation into postfix notation (Reverse Polish Notation, RPN). It relies on two structures:

- A **stack** for operators/functions
- An **output queue** for the final postfix expression

K. Algorithm Steps

1) Initialize an empty stack and output queue.
2) For each token:
   a) Numbers → add to output.
   b) Functions → push to stack.
   c) Operators → pop higher/equal precedence operators from stack to output (respecting associativity), then push current operator.
   d) "(" → push to stack.
   e) ")" → pop to output until matching "(" is found, discard brackets.
3) After processing, pop remaining operators to output.

Algorithm 2 Shunting Yard Algorithm (Infix to Postfix)

```
function SHUNTINGYARD(expr)
    Initialize empty stack operators
    Initialize empty string output
    for each character c in expr do
        if c is a number or '.' then
            Append c to output until number ends
            Append space to output
        else if c is a function then
            Push c onto operators
        else if c = ( then
            Push c onto operators
            Increment unmatched_brackets
        else if c =) then
            while Top of operators ≠ ( do
                Pop from operators and append to output
            end while
            Pop '(' from operators
            Decrement unmatched_brackets
            if Top of operators is a function then
                Pop function and append to output
            end if
        else if c is an operator in {+, −, ∗, /, ^, !} then
            while operators not empty and precedence(Top of operators) ≥ precedence(c) do
                Pop from operators and append to output
            end while
            Push c onto operators
        end if
    end for
    while operators not empty do
        Pop from operators and append to output
    end while
    return output
end function
```

L. Operator Precedence

M. Example

For the expression $3 + 4 * 2/(1 - 5)$, the Shunting Yard Algorithm yields the postfix form:

$$3\ 4\ 2\ *\ 1\ 5\ -\ /\ + \tag{19}$$

Algorithmic Steps :

N. Reverse Polish Notation (RPN)

Reverse Polish Notation, also known as postfix notation, is a way of writing mathematical expressions in which operators follow their operands. Unlike infix notation (e.g., $3 + 4$), RPN eliminates the need for parentheses by unambiguously encoding operator precedence.

- Example (infix): $3 + 4$
- Equivalent RPN: $3\ 4\ +$

This simplicity makes RPN especially suitable for evaluation using a stack-based algorithm.

TABLE II
RK4 UPDATE EQUATIONS FOR SINGLE-VARIABLE FUNCTIONS

Function	k_1	k_2	k_3	k_4	y_{n+1}
Arcsine $\arcsin(x)$	$\frac{h}{\sqrt{1-x_n^2}}$	$\frac{h}{\sqrt{1-(x_n+\frac{h}{2})^2}}$	$\frac{h}{\sqrt{1-(x_n+\frac{h}{2})^2}}$	$\frac{h}{\sqrt{1-(x_n+h)^2}}$	$y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
Arctangent $\arctan(x)$	$\frac{h}{1+x_n^2}$	$\frac{h}{1+(x_n+\frac{h}{2})^2}$	$\frac{h}{1+(x_n+\frac{h}{2})^2}$	$\frac{h}{1+(x_n+h)^2}$	$y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
Natural Log $\ln(x)$	$\frac{h}{x_n}$	$\frac{h}{x_n+\frac{h}{2}}$	$\frac{h}{x_n+\frac{h}{2}}$	$\frac{h}{x_n+h}$	$y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
Power $y = x^w$	$h \cdot \frac{w}{x_n} y_n$	$h \cdot \frac{w}{x_n+\frac{h}{2}} \left(y_n + \frac{k_1}{2}\right)$	$h \cdot \frac{w}{x_n+\frac{h}{2}} \left(y_n + \frac{k_2}{2}\right)$	$h \cdot \frac{w}{x_n+h} (y_n + k_3)$	$y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

TABLE III
OPERATOR PRECEDENCE AND ASSOCIATIVITY

Operator	**Precedence**	**Associativity**
$\vee$ (power)	4	Right
$*$, $/$	3	Left
$+$, $-$	2	Left

TABLE IV
SHUNTING YARD EXAMPLE FOR $3 + 4 * 2/(1 - 5)$

Token	**Action**	**Stack**	**Output**
3	Add to output		3
+	Push to stack	+	3
4	Add to output	+	3 4
*	Push to stack	+ *	3 4
2	Add to output	+ *	3 4 2
/	Pop *, push /	+ /	3 4 2 *
(	Push to stack	+ / (	3 4 2 *
1	Add to output	+ / (	3 4 2 * 1
-	Push to stack	+ / (-	3 4 2 * 1
5	Add to output	+ / (-	3 4 2 * 1 5
)	Pop until (	+ /	3 4 2 * 1 5 -
End	Pop stack		3 4 2 * 1 5 - / +

O. Evaluating RPN

Evaluation of RPN uses a stack:

1) Scan tokens left to right.
2) Push numbers on the stack.
3) For operators, pop required operands, apply operation, push result.

For the above example, evaluating the postfix expression gives the result 1 as expected .

P. Complexity and Applications

The algorithm runs in $O(n)$ time, where n is the number of tokens, since each token is processed once. Combined with RPN evaluation, it provides an efficient method for handling expressions in calculators, compilers, and symbolic computation systems.

Q. Function Handling

The Shunting Yard Algorithm can also be used to implement functions such as $\sin(x)$, $\cos(x)$, and others. Function handling follows these rules:

1) When a function token is encountered, it is pushed onto the operator stack.
2) Arguments (possibly separated by commas) are processed as sub-expressions in the usual way.
3) When the closing parenthesis ")" is reached:
   - Operators are popped from the stack to the output queue until the matching "(" is found.
   - The function token itself is then moved from the stack to the output queue.

As a result, function calls are correctly represented in postfix form. For example:

$$\sin(x) \quad \rightarrow \quad x \ \sin, \tag{20}$$

R. Conclusion

The Shunting Yard Algorithm, together with RPN evaluation, ensures efficient parsing and evaluation of mathematical expressions. It respects operator precedence, associativity, and bracket handling, making it a cornerstone in expression processing across computing applications.

S. Code Repository

The complete C implementations of the algorithms discussed in this paper are available at: https://github.com/gadepall/calculator/tree/main/codes

```
codes/
|-- ShuntingYard.c
|-- inv_sq_root.c
|-- inv_trig_func.c
|-- log.c
|-- power_func.c
|-- trig_func.c
```

Listing 1. Repository Structure in codes/

II. Hardware Implementation

A. Hardware Required

TABLE V
MATERIALS REQUIRED FOR THE SCIENTIFIC CALCULATOR

Quantity	Component
25	Pushbuttons
1	LCD 16×2
1	Arduino Uno
–	Jumper wires
1	Potentiometer

- A button matrix for user input.
- An Arduino Uno microcontroller to process inputs and execute calculations.
- A 16×2 LCD connected to Arduino for displaying results.
- Connections between the button matrix, LCD, and Arduino Uno as shown in Fig. 2.

B. Circuit Connections

TABLE VI
ARDUINO TO LCD PIN CONNECTIONS

Arduino Pin	LCD Pin	LCD Label	Description
GND	1	GND	Ground
5V	2	Vcc	Power Supply
GND	3	Vee	Contrast Control
D2	4	RS	Register Select
GND	5	R/W	Read/Write (fixed to Write)
D3	6	EN	Enable
D4	11	DB4	Data Bus (4-bit mode)
D5	12	DB5	Data Bus (4-bit mode)
D6	13	DB6	Data Bus (4-bit mode)
D7	14	DB7	Data Bus (4-bit mode)
5V	15	LED+	Backlight Anode
GND	16	LED-	Backlight Cathode

C. Button Matrix

The button matrix is a grid of push-buttons arranged in rows and columns. It allows multiple buttons to be connected to the microcontroller using fewer pins.

Working Principle:

- The Arduino scans the matrix by activating each row (setting it LOW) one at a time while reading the columns.
- When a button is pressed, it completes the circuit between its corresponding row and column.
- By identifying the active row and column, the Arduino determines which button was pressed.
- Example: If the first column is active and a press is detected on the second row, the button pressed is the second button in the first column.

This reduces the number of pins required to implement a calculator with 25 buttons.

TABLE VII
KEY ASSIGNMENTS IN MODE 1 WITH ARDUINO PIN MAPPING

	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	0	1	2	3	4
Row 2	5	6	7	8	9
Row 3	+	-	×	÷	sin(
Row 4	cos(	tan(	$e^($	ln(	Clear
Row 5	Backspace	.	=	Mode shift	π

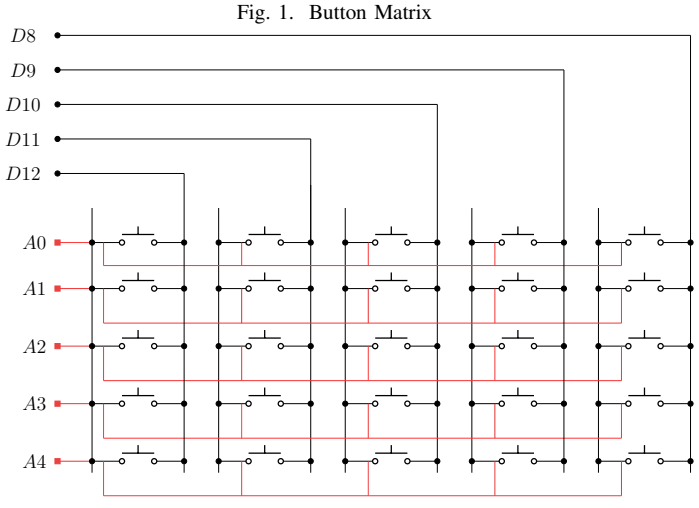
TABLE VIII
KEY ASSIGNMENTS IN MODE 2 WITH ARDUINO PIN MAPPING

	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	0	1	2	3	4
Row 2	5	6	7	8	9
Row 3	(	)	x^y	fact(	arcsin(
Row 4	arccos(	arctan(	mod	$\log_{10}($	Clear
Row 5	Backspace	.	=	Mode shift	π

TABLE IX
KEYPAD ROW/COLUMN CONNECTIONS TO ARDUINO PINS

Keypad Line	Arduino Pin
Row 1	D8
Row 2	D9
Row 3	D10
Row 4	D11
Row 5	A3
Col 1	D12
Col 2	D13
Col 3	A0
Col 4	A1
Col 5	A2

Fig. 1. Button Matrix



D. Schematic Circuit

The schematic for circuit connections is shown below,

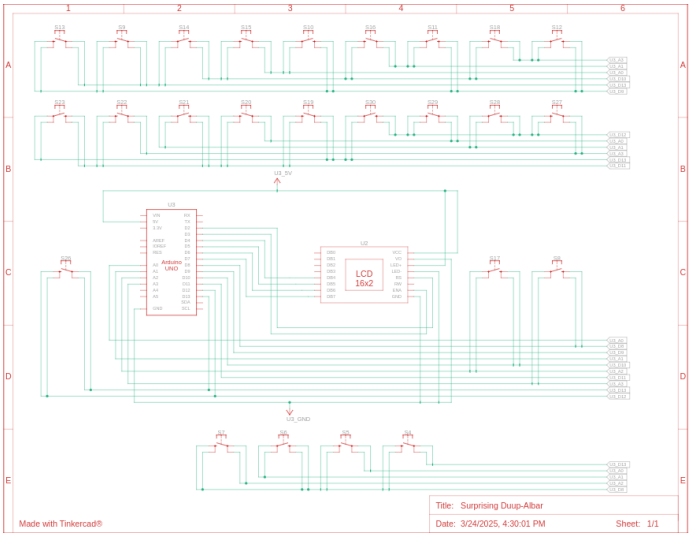


Fig. 2. Schematic of Circuit.

E. Hardware Circuit

The hardware circuit of calculator,

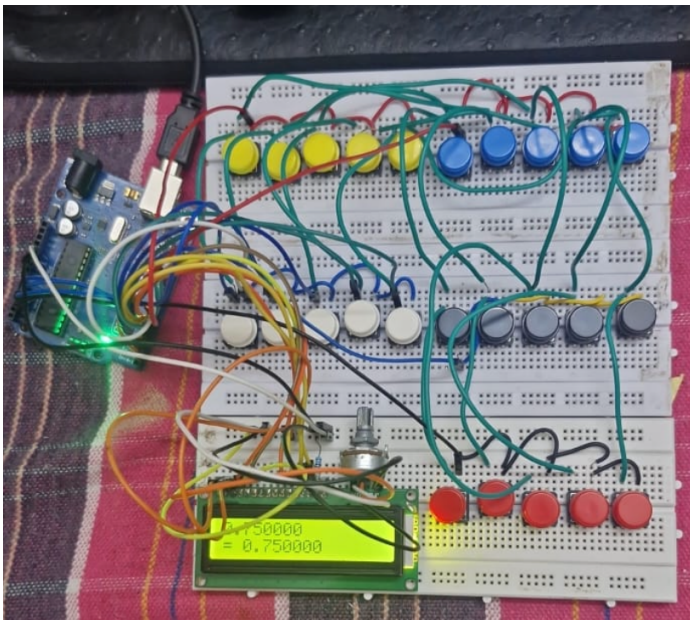


Fig. 3. circuit

APPENDIX A
RUNGE–KUTTA METHOD (RK4)

The RK4 method is a numerical technique to solve

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0 \tag{21}$$

with step size h. The next value is calculated as:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{22}$$

where

$$k_1 = hf(x_n, y_n), \tag{23}$$
$$k_2 = hf(x_n + h/2, y_n + k_1/2), \tag{24}$$
$$k_3 = hf(x_n + h/2, y_n + k_2/2), \tag{25}$$
$$k_4 = hf(x_n + h, y_n + k_3). \tag{26}$$

Quake III Algorithm: The method obtains an initial approximation by manipulating the IEEE 754 floating-point representation of x, then refines it using Newton-Raphson iterations.

1) Bit manipulation with a "magic constant" $(0x5f3759df)$ produces an initial guess y_0.
2) A single Newton–Raphson iteration dramatically improves accuracy.
3) An optional second iteration yields nearly full precision.

Newton–Raphson Refinement: We want to approximate

$$y = x^{-\frac{1}{2}}. \tag{27}$$

Define

$$f(y) = \frac{1}{y^2} - x, \qquad f'(y) = -\frac{2}{y^3}. \tag{28}$$

Applying Newton–Raphson,

$$y_{k+1} = y_k - \frac{f(y_k)}{f'(y_k)} \tag{29}$$

$$= y_k - \frac{\frac{1}{y_k^2} - x}{-\frac{2}{y_k^3}} \tag{30}$$

$$= y_k + \frac{y_k}{2}\left(1 - x y_k^2\right) \tag{31}$$

$$= y_k \left(\tfrac{3}{2} - \tfrac{1}{2} x y_k^2\right). \tag{32}$$

Thus one Newton iteration is simply

$$y \leftarrow y\left(1.5 - 0.5\, x\, y^2\right). \tag{33}$$

REFERENCES

[1] B. S. Grewal, *Higher Engineering Mathematics*, 43rd ed. New Delhi, India: Khanna Publishers, 2014.
[2] G. V. V. Sharma,C Programming in Middle School
[3] E. Kreyszig, *Advanced Engineering Mathematics*, 10th ed. Hoboken, NJ, USA: John Wiley & Sons, 2011.
[4] National Council of Educational Research and Training (NCERT), *Mathematics: Textbook for Class XII*, Part 1 and 2, New Delhi, India: NCERT, 2006.