## 

**Department of Computer Science and Engineering**

**BRAINWARE UNIVERSITY**

**398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata – 700125**

**APPENDIX 1**

Special Calculator

It is a special type of calculator where we can use both “Simple Calculator” and “Scientific Calculator”

Submitted by: Aditya Saha

Aditya Saha  
Subhajit Dhar

Sampad Dey

Sayan Mandal

Arijit Malakar

#### In partial fulfillment for the award of the degree of

Diploma in CSE

## 

**Department of Computer Science and Engineering**

**BRAINWARE UNIVERSITY**

**398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata – 700125**

Special Calculator

Submitted by

Aditya Saha

***in partial fulfillment for the award of the degree***

of

## Diploma in Computer Science and Engineering

in

Computer Science and Engineering



**Department of Computer Science and Engineering**

**BRAINWARE UNIVERSITY**

**398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata – 700125**

# APPENDIX 2

**Bonafide Certificate –** The Bonafide Certificate shall be in double line spacing using Font Style Calibri and Font Size 14**.** The certificate shall carry the supervisor’s signature and shall be followed by the supervisor’s name, academic designation (not any other responsibilities of administrative nature), department and full address of the institution where the supervisor has guided the student.

The term **‘SUPERVISOR’** must be typed in capital letters between the supervisor’s name and academic designation.

##### [A specimen of the BONAFIDE CERTIFICATE]

**[**Computer Science and Engineering**]**



**Department of Computer Science and Engineering**

**BRAINWARE UNIVERSITY**

**398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata – 700125**

**BONAFIDE CERTIFICATE**

Certified that this project report “**Special Calculator**” is the bona fide work of

“**Aditya Saha,**

**Subhajit Dhar,**

**Sampad Dey,**

**Sayan Mandal,**

**Arijit Malakar”**

who carried out the project work under my supervision.

##### SIGNATURE SIGNATURE

**Dr. Avijit Kumar Mr. Pranab Gharai**

##### HEAD OF THE DEPARTMENT SUPERVISOR

Computer Science and Engineering Computer Science and Engineering

Department of Computer Science and Engineering

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata –700125



**Department of Computer Science and Engineering**

**BRAINWARE UNIVERSITY**

**398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata – 700125**

# APPENDIX 3

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **Chapter** | **Title** | **Page No.** |
|  | **ABSTRACT** | **i** |
|  | **LIST OF SYMBOLS** | **ii** |
| **1.** | **INTRODUCTION** | **1** |
| **2.** | Literature Review | **1** |
| **3.** | Theory, Methodology, Materials & Methods | **2** |
| **4.** | **Results, Analysis & Discussions** | **3** |
| **5.** | **Conclusion, Future Scope, Limitations** | **4** |
| **6.** | Appendices | **5** |



**Department of Computer Science and Engineering**

**BRAINWARE UNIVERSITY**

**398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata – 700125**

# APPENDIX 4

**List of References** –The reference material should be listed in the alphabetical order of the first author. The name of the author/authors should be immediately followed by the year and other details.

A typical illustrative list given below relates to the citation example quoted above.

##### REFERENCES

1. Aritikula, S. and Nandi, S. (1994) ‘Transport Phonomena of Sm Sel – X Asx’, Pramana – Journal of Physics Vol.42, No.1, pp.421-425.
2. Bernard, R.W. and Keiger, C. (1980) ‘Applications of Convolution Theory Operators to Problems in Univalent Function Theory’, Michigan Mach, J., Vol.27, pp.81–94.
3. Shaw, K.G. and Mckay, N.D. (1984) ‘Open Loop Maximum Time Control of Mechanical Motoring and its Applications’, Proc. Amer. Contr. Conf., San Diego, CA, pp. 1231-1236.

## 1. Abstract

This project details the development of a **modern, interactive web-based calculator** application implemented using **pure JavaScript, HTML, and Tailwind CSS**. The application features a dynamic interface with two distinct modes: a **Simple Calculator** for basic arithmetic operations (+, -, \*, /) and a **Scientific Calculator** offering advanced functions, including trigonometric (sin, cos, tan), logarithmic (ln, $\log\_{10}$), exponential ($e^x$), and constants ($\pi$). The core functionality relies on the **handleInput** function to process user input and dynamic expression parsing, utilizing a custom **parseScientificExpression** function and the **new Function()** constructor for secure evaluation. The design emphasizes **responsive UI** and **user experience** through state management, a visual history display, and subtle CSS animations for feedback. The project demonstrates proficiency in **front-end development**, **DOM manipulation**, and the implementation of a functional state machine in a web context.

## 2. List of Symbols, Abbreviations and Nomenclature

|  |  |
| --- | --- |
| **Abbreviation / Symbol** | **Description** |
| **JS** | JavaScript |
| **HTML** | HyperText Markup Language |
| **CSS** | Cascading Style Sheets |
| **DOM** | Document Object Model |
| **UI** | User Interface |
| **PI** | Mathematical constant $\pi$ (approximately 3.14159) |
| **powE** | Mathematical function $e^x$ (exponential function) |
| **sin()** | Sine trigonometric function (expects degrees in this implementation) |
| **ln()** | Natural logarithm function ($\log\_e$) |
| **log10()** | Logarithm base 10 function |
| **factorial()** | Factorial function ($n!$) |

## 3. Chapters

### Chapter 1: Introduction

This chapter provides an overview of the calculator project, setting the context for its development.

* **1.1 Background:** Calculators are fundamental tools in mathematics, engineering, and daily life. The transition to web-based applications allows for accessible, platform-independent, and feature-rich digital tools.
* **1.2 Project Goal:** The primary objective was to design and implement a **versatile and responsive calculator** that combines the utility of a basic arithmetic calculator with the power of a scientific one, all within a modern web interface.
* **1.3 Scope:** The project includes the development of the user interface (UI) using HTML and Tailwind CSS, the state management logic in JavaScript, and the implementation of core arithmetic and a suite of scientific functions.

### Chapter 2: Literature Review

This section reviews key technologies and existing solutions that influenced the project's architecture.

* **2.1 Web Development Stacks:** The choice of **HTML, CSS (Tailwind CSS), and JavaScript** represents the core stack for modern front-end development, offering high compatibility and performance.
* **2.2 Calculator Logic and Parsing:** Reviewing the logic of existing software calculators showed the necessity of robust **expression parsing**. The chosen approach uses regular expressions in the parseScientificExpression function to transform scientific notation (e.g., sin(x)) into valid JavaScript Math object calls (e.g., Math.sin(...)).
* **2.3 UI/UX Considerations:** Modern web application design emphasizes responsiveness and visual feedback. The use of Tailwind CSS facilitated rapid styling and ensured a mobile-friendly layout. Animations, like the result-pop effect, were incorporated to improve user feedback.

### Chapter 3: Theory, Methodology, Materials & Methods

This chapter outlines the technical approach, architecture, and core components used in the project.

#### 3.1 Materials (Technologies)

* **HTML:** Used for the structure of the application (index.html).
* **JavaScript (ES6+):** The primary language for application logic and state management (script.js).
* **CSS (Tailwind CSS):** Utilized for utility-first styling, ensuring a clean and responsive design (style.css and CDN in index.html).

#### 3.2 Methodology

The project followed a modular, component-based development approach:

1. **State Management:** The global **calculatorState** object tracks the current mode, display content, and history.
   * const calculatorState = { currentMode: 'menu', display: '0', history: '', };
2. **Rendering Functions:** Functions like **renderMenu()**, **renderHeader()**, **renderDisplay()**, and **renderKeypad()** are responsible for dynamically generating and updating the HTML structure based on the current mode.
3. **Input Handling:** The **handleInput(input)** function is the central event handler, responsible for:
   * **Clear (C)** and **Equals (=)** operations.
   * Handling number and operator concatenation.
   * Managing the **sign change (+/-)** and **decimal point (.)** logic.

#### 3.3 Core Theory and Functions

* **Expression Evaluation:** The core calculation relies on the following sequence:
  1. The expression string is taken from calculatorState.display.
  2. For scientific mode, the string is pre-processed by **parseScientificExpression()** to convert user-friendly function calls (e.g., sin(30)) into executable JavaScript (e.g., Math.sin((30) \* Math.PI / 180)).
  3. The final, safe expression string is evaluated using the **new Function(\return ${expression}`)()`** approach for execution.
* Trigonometric Conversion: The code implements trigonometric functions assuming input in degrees. This requires an internal conversion to radians before calling JavaScript's built-in Math functions, as seen in:  
    
  $$\text{JS Expression} = \text{Math.sin}((\text{Input Value}) \cdot \frac{\pi}{180})$$
* **Factorial Function:** A custom function, **factorial(n)**, is implemented to handle the $n!$ operation, ensuring correct handling of negative and zero inputs:  
  JavaScript  
  function factorial(n) {  
   if (n < 0) return NaN;  
   if (n === 0) return 1;  
   // ... loop for calculation  
  }

### Chapter 4: Results, Analysis & Discussions

#### 4.1 Results

The developed application successfully provides three primary views:

* **Menu View:** Allows the user to select between **Simple** and **Scientific** modes.
* **Simple Calculator View:** A 4-column layout for basic arithmetic.
* **Scientific Calculator View:** A 5-column layout providing access to advanced functions.

All results are displayed in the dedicated display area, with the full expression visible in the history line (DOMElements.historyText) upon calculation.

#### 4.2 Analysis

* **Functionality:** All standard and scientific operations implemented (e.g., sin(), log10(), e^x, factorial()) function correctly, with appropriate handling for degree-to-radian conversion.
* **Robustness (Error Handling):** The try...catch block in handleInput successfully captures syntax or mathematical errors during evaluation, displaying **"Error"** to the user instead of crashing the application.
* **UI/UX:** The separation of concerns between JavaScript logic and Tailwind CSS styling resulted in a visually appealing and highly maintainable codebase. The use of grid layouts ensures responsiveness across different screen sizes.

#### 4.3 Discussions

The decision to use **new Function()** for expression evaluation, while powerful, requires careful input sanitization (which is implicitly handled by the restricted input logic of a button-based UI) to prevent potential security issues in a general-purpose evaluator. For this closed-loop calculator, it offers the simplest and most effective way to evaluate dynamically constructed mathematical strings. The use of Regular Expressions in parseScientificExpression proved highly efficient for transforming the user-entered function syntax into valid JS expressions.

### Chapter 5: Conclusion, Future Scope, Limitations

#### 5.1 Conclusion

The project successfully delivered a fully functional, two-mode web calculator. It validates the effectiveness of using JavaScript for application logic and state management alongside utility-first CSS for creating a modern, high-quality user interface. The implementation demonstrates a solid understanding of front-end architecture, dynamic DOM manipulation, and mathematical expression handling.

#### 5.2 Future Scope

Potential enhancements for future development include:

* **Support for Degrees/Radians Toggle:** Allowing users to switch between expecting degree and radian inputs for trigonometric functions.
* **Memory Functions (M+, M-, MR):** Implementing a persistent memory feature.
* **Graphing Mode:** Integrating a library (e.g., Plotly or D3.js) to plot single-variable functions entered by the user.
* **Enhanced Error Feedback:** Providing more specific error messages (e.g., "Syntax Error" or "Math Error") instead of a generic "Error."

#### 5.3 Limitations

* **Expression Complexity:** The current parsing is limited to the defined functions and basic operators; it does not handle complex chained functions or implicit multiplication (e.g., $2\pi$).
* **Security of new Function():** While mitigated by the button-only input, for a broader application, a dedicated mathematical expression parser would be safer and more robust than relying on dynamic code execution.

## 4. Appendices

### Appendix A: Key JavaScript Code Snippets

JavaScript

// A.1 State and DOM Elements  
const calculatorState =   
 {  
 currentMode: 'menu',  
 display: '0',  
 history: '',  
 };  
const DOMElements = {   
 // ...  
 };  
  
// A.2 Scientific Expression Parsing  
function parseScientificExpression(expr) {  
 let parsedExpr = expr.replace(/x/g, '\*');  
  
 // ... trigonometric conversion from degrees to radians  
 parsedExpr = parsedExpr.replace(/sin\(([^)]+)\)/g, (match, p1) => `Math.sin((${p1}) \* Math.PI / 180)`);  
  
 // ... evaluation using new Function()  
 return parsedExpr;  
 }  
  
// A.3 Main Input Handler (Extract)  
if (input === '=') {  
 try {  
 let expression = calculatorState.currentMode === 'scientific'  
 ? parseScientificExpression(currentDisplay)  
 : currentDisplay.replace(/x/g, '\*');  
  
 const result = new Function(`return ${expression}`)();  
  
 // ... update state and display  
 } catch (e) {  
 calculatorState.display = 'Error';  
 }  
 updateDisplay();  
 return;  
}