CALCULATOR

Submitted by

SHAIK.FARHEEN

INDEX

ABSTRACT

This project presents the design and implementation of a Digital Calculator, a user-friendly tool developed to perform basic arithmetic and scientific operations efficiently. The calculator is capable of executing fundamental functions such as addition, subtraction, multiplication, and division, as well as advanced features like square root, exponentiation, and trigonometric calculations. Developed using [insert technology/language used, e.g., Python with Tkinter, Java, or embedded systems], the project emphasizes accuracy, speed, and a simple graphical user interface to enhance user interaction. The digital calculator is designed to support both educational and practical applications, providing users with a reliable tool for solving mathematical problems. The project also explores logical flow, algorithm optimization, and error handling to ensure robustness and efficiency. Overall, this digital calculator serves as a foundation for understanding computational logic and user interface design in software/hardware development

PROBLEM IDENTIFICATION

CHALLENGES IN CALCULATOR

- **User Interface Design**
  Designing a clear, intuitive, and responsive user interface was challenging, especially when arranging buttons, display areas, and ensuring compatibility across different screen sizes (in GUI or web projects).
- **Handling Invalid Inputs**
  Ensuring the calculator correctly handled errors such as division by zero, square roots of negative numbers, or invalid syntax required robust input validation and error handling mechanisms.
- **Logic Implementation**
  Implementing the correct order of operations (BODMAS/PEMDAS) and managing complex expressions posed a significant logic challenge, especially when supporting chained operations or parentheses.

PROPOSED SOLUTION:

To overcome these challenges, our **CALCULATOR case study** focuses on developing an **AI- powered opponent** using the **Shunting Yard + Postfix Evaluation algorithm**. This ensures:

- **User Interface Design**

Used a grid layout system (in GUI frameworks like Tkinter, Java Swing, or HTML/CSS Flex/Grid) to neatly organize buttons and display elements. User testing was done to ensure ease of use and visual clarity.

- **Handling Invalid Inputs**

Implemented input validation using try-except blocks (in Python) or exception handling (in Java/C++) to gracefully manage errors such as division by zero or invalid expressions, displaying appropriate messages.

- **Logic Implementation**

Developed a custom expression parser or used built-in eval() functions carefully (with input sanitization) to handle the correct order of operations. Where applicable, stack-based logic was used to implement expression evaluation.

TECHNICAL PROBLEM BREAKDOWN

1. Expression Parsing
   - Problem: Handling multi-operator expressions (e.g., 5 + 2 * (3 - 1)).
   - Cause: Difficulty in applying operator precedence and associativity rules.
   - Impact: Produces incorrect results or crashes.
   - Solution: Implement a parsing algorithm (e.g., Shunting Yard) or use built-in safe evaluation functions with syntax validation.

---

2. Input Validation
   - Problem: Invalid user inputs (e.g., 2++3, /0, sqrt(-1)).
   - Cause: No input sanitization or lack of error-checking logic.
   - Impact: Crashes or shows misleading outputs.
   - Solution: Add error handling using try-except (Python), try-catch (Java), or conditional checks.

---

3. Floating-Point Precision
   - Problem: Inaccurate or very long decimal results (e.g., 0.1 + 0.2 shows 0.30000000000000004).
   - Cause: Internal floating-point representation limits.
   - Impact: Confuses users, reduces trust in accuracy.
   - Solution: Use rounding methods or the Decimal class (Python), BigDecimal (Java) for precise calculations.

.

OBJECTIVE SETTING

1. Primary Objective
   - To design and develop a digital calculator capable of performing basic arithmetic and scientific operations with accuracy, efficiency, and user-friendly interaction.

---

2. Specific Objectives
   1. Perform Core Arithmetic Operations
      - Implement addition, subtraction, multiplication, and division functionalities.
   2. Implement Scientific Functions (Optional)
      - Integrate operations such as square root, power, trigonometric functions (sin, cos, tan), and logarithms (depending on scope).
   3. Develop an Intuitive User Interface
      - Create a clear and accessible layout using a suitable GUI framework or hardware interface (e.g., LCD + keypad for Arduino).
   4. Ensure Input Validation and Error Handling
      - Prevent invalid operations (e.g., divide by zero, square root of negative number) using proper checks and feedback mechanisms.
   5. Follow Operator Precedence Rules
      - Accurately evaluate complex expressions by implementing or integrating algorithms that respect mathematical order of operations.
   6. Optimize for Performance and Accuracy
      - Deliver precise results with appropriate rounding/formatting while maintaining fast response time.
   7. Develop Modular and Maintainable Code
      - Write clean, well-documented, and modular code to allow easy debugging, testing, and future enhancements.
   8. Ensure Cross-Platform or Device Compatibility
      - If applicable, ensure the calculator functions across different platforms or works reliably on hardware devices.

Timeline:

1. Project planning and requirements gathering: 2 days
2. Game design and development: 10 days
3. Testing and debugging: 5 days
4. Deployment and maintenance: 3 days

Deliverables:

1. A fully functional Calculator.
2. A user manual and instructions for calculator.
3. A technical report detailing the calculator’s architecture, design, and development.
4. A testing report detailing the calculator's functionality, usability, and performance.

KEY WORDS

- Digital Calculator
- Arithmetic Operations
- Scientific Calculator
- Expression Evaluation
- User Interface (UI)
- GUI Programming
- Input Validation
- Error Handling
- Operator Precedence (BODMAS/PEMDAS)
- Event Handling
- Programming Logic
- Expression Parser
- Floating Point Precision
- Stack-Based Evaluation
- Shunting Yard Algorithm
- Modular Design
- Software Development
- Code Optimization
- Embedded System (if hardware-based)
- Arduino (if applicable)
- Python / Java / C++ / JavaScript (based on your languag

CODE

```
import tkinter as tk
import math

def click(event):
    current = entry.get()
    button_text = event.widget.cget("text")

    try:
        if button_text == "=":
            result = eval(current)
            entry.delete(0, tk.END)
            entry.insert(tk.END, str(result))

        elif button_text == "C":
            entry.delete(0, tk.END)

        elif button_text == "√":
            result = math.sqrt(float(current))
            entry.delete(0, tk.END)
            entry.insert(tk.END, str(result))

        elif button_text == "^":
            entry.insert(tk.END, "**")

        elif button_text == "π":
            entry.insert(tk.END, str(math.pi))

        elif button_text == "e":
            entry.insert(tk.END, str(math.e))

        elif button_text == "sin":
            result = math.sin(math.radians(float(current)))
            entry.delete(0, tk.END)
            entry.insert(tk.END, str(result))

        elif button_text == "cos":
```

```
            result = math.cos(math.radians(float(current)))
            entry.delete(0, tk.END)
            entry.insert(tk.END, str(result))

        elif button_text == "tan":
            result = math.tan(math.radians(float(current)))
            entry.delete(0, tk.END)
            entry.insert(tk.END, str(result))

        elif button_text == "log":
            result = math.log10(float(current))
            entry.delete(0, tk.END)
            entry.insert(tk.END, str(result))

        elif button_text == "ln":
            result = math.log(float(current))
            entry.delete(0, tk.END)
            entry.insert(tk.END, str(result))

        else:
            entry.insert(tk.END, button_text)

    except Exception as e:
        entry.delete(0, tk.END)
        entry.insert(tk.END, "Error")

# GUI setup
root = tk.Tk()
root.title("Scientific Calculator")
root.geometry("400x550")

entry = tk.Entry(root, font="Arial 24")
entry.pack(fill=tk.BOTH, ipadx=8, pady=10)

# Buttons grid
buttons = [
    ['7', '8', '9', '/', '√'],
    ['4', '5', '6', '*', '^'],
```

```
    ['1', '2', '3', '-', 'π'],
    ['0', '.', '=', '+', 'e'],
    ['sin', 'cos', 'tan', 'log', 'ln'],
    ['C']
]

for row in buttons:
    frame = tk.Frame(root)
    for label in row:
        btn = tk.Button(frame, text=label, font="Arial 18", width=6, height=2)
        btn.pack(side=tk.LEFT, padx=5, pady=5)
        btn.bind("<Button-1>", click)
    frame.pack()

root.mainloop()
```

OUTPUT