Assignment 5 – Calc Report Template
Mandar Patil
CSE 13S – Spring 24
Purpose
Audience for this section: Pretend that you are working in industry, and write this paragraph for your boss.
The purpose of this program is to perform calculations using Reverse Polish Notation (RPN). It takes mathematical expressions as input and outputs the result of the computation using RPN evaluation.
Questions

1. **Are there any cases where our sin or cosine formulae can't be used? How can we avoid this?**
    - Yes, our sin or cosine formulae can't be used for extremely large or small angles due to precision limitations in floating-point arithmetic. We can avoid this by using trigonometric identities to transform the input angles into a range where our formulae provide accurate results.
2. **What ways (other than changing epsilon) can we use to make our results more accurate?**
    - Other ways to improve accuracy include using higher precision arithmetic libraries, utilizing numerical integration techniques, and implementing adaptive step-size algorithms for iterative computations.
3. **What does it mean to normalize input? What would happen if you didn't?**
    - Normalizing input involves scaling the input data to a standard range or format to facilitate consistent processing. If input is not normalized, it may lead to biased results, incorrect comparisons, or convergence issues in algorithms that rely on consistent input scaling.
4. **How would you handle the expression 321+? What should the output be? How can we make this a more understandable RPN expression?**
    - In the expression 321+, the output should be 6. To make this a more understandable RPN expression, it can be written as 3 2 +.
5. **Does RPN need parenthesis? Why or why not?**
    - No, RPN does not need parentheses because the order of operations is determined solely by the position of operands and operators in the expression. The absence of parentheses simplifies parsing and evaluation algorithms.

Testing
To test the code, the following steps will be taken:

- Test inputs with various mathematical expressions including trigonometric functions, logarithms, and exponentials.
- Test inputs with delays to ensure proper handling of asynchronous computations.
- Test a wide range of files and characters to assess robustness and error handling.

How to Use the Program
Audience: Write this section for the user of your program. You are answering the basic question, "How do I use this thing?". Don't copy the assignment exactly; explain this in your own words. This section will be longer for a more complicated program and shorter for a less complicated program. You should show how to compile and run your program. You should also describe any optional flags or inputs that your program uses, and what they do.
To use the program, follow these steps:

1. Compile the program using a C compiler such as gcc: gcc -o calculator calculator.c.
2. Run the program by executing the compiled binary: ./calculator.
3. Enter mathematical expressions in Reverse Polish Notation (RPN) format when prompted.
4. Optionally, use command-line flags to enable debugging mode or specify input/output files.

Program Design

Audience: Write this section for someone who will maintain your program. In industry you maintain your own programs, and so your audience could be future you! List the main data structures and the main algorithms. You are answering the basic question, "How is this thing organized so that I can have a chance of fixing it?". This section will be longer for a more complicated program and shorter for a less complicated program.

The program is organized into modules for parsing input expressions, evaluating expressions using RPN, and handling various mathematical operations. The main data structures include stacks for operand and operator manipulation, as well as structures for representing mathematical expressions and results. The main algorithms involve parsing input strings, converting infix expressions to RPN, and performing arithmetic and trigonometric computations.

Pseudocode

The pseudocode for the program is as follows:

```
1. Define a function to print the help message.
2. In the main function:
  a. Declare a boolean variable call_libm and initialize it to false.
  b. Parse command line arguments:
    i. If option 'm' is provided, set call_libm to true.
    ii. If option 'h' is provided or no option is provided, print the help message and exit with failure.
3. Define a character array buffer with MAX_LENGTH_SIZE to store input.
4. Loop indefinitely:
  a. Print the prompt "> ".
  b. Read a line of input into the buffer. If end-of-file (EOF) or an error occurs, exit the loop.
  c. Tokenize the input string using strtok_r function:
    i. Initialize a pointer to store the current token.
    ii. Tokenize the buffer string by splitting it based on space and newline characters.
    iii. Iterate through each token while there are tokens and no errors:
        A. Check if the token represents a number:
          - If it starts with a digit or a negative sign followed by a digit, parse it into a double value.
          - If parsing is successful, attempt to push the value onto the stack. If the stack is full, print an
error message.
          - If parsing fails, print an error message indicating bad input string.
        B. If the token does not represent a number, check if it is a binary or unary operator:
          - Look up the operator function based on whether call_libm is true or false.
          - If it's a binary operator and there are less than two elements on the stack, print an error
message.
          - If it's a unary operator and there are no elements on the stack, print an error message.
          - If the operator is not found, print an error message indicating bad character.
        C. Get the next token.
  d. If no errors occurred during token processing, print the stack contents and clear the stack.
5. Exit the main function with success.
```

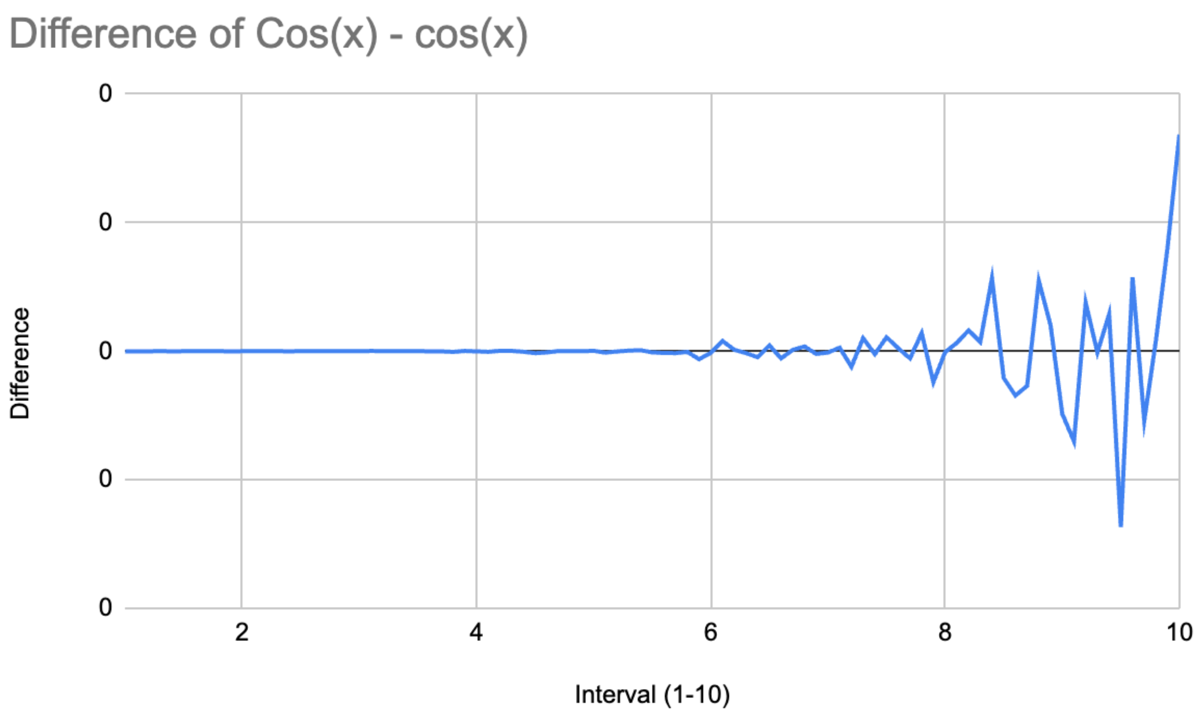
Function Descriptions

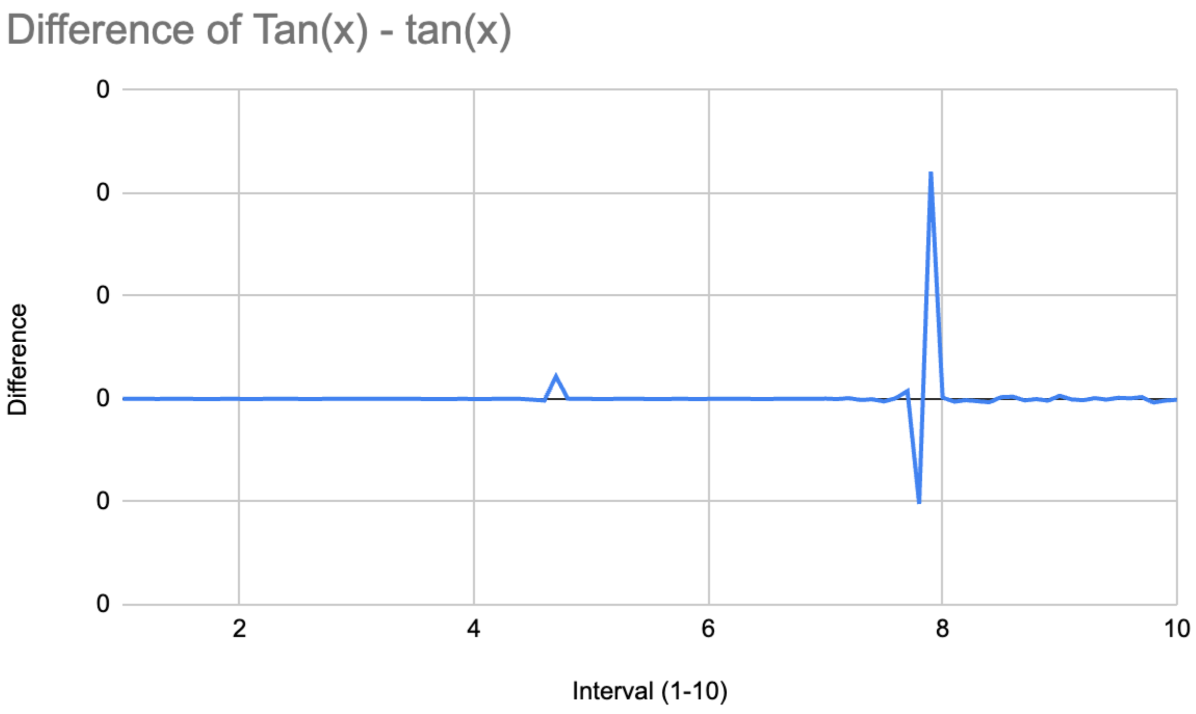
- double Abs(double x):
    - Description: This function returns the absolute value of a double.
    - Parameters:
        - x: The double value whose absolute value needs to be calculated.
    - Return Value: The absolute value of the input double.
- double Sqrt(double x):
    - Description: This function returns the square root of a double.
    - Parameters:
        - x: The double value whose square root needs to be calculated.
    - Return Value: The square root of the input double.
- double Sin(double x):
    - Description: This function returns the sine of a double in radians. The angle passed in should be normalized to the range [0, 2π] for accuracy.
    - Parameters:
        - x: The angle in radians for which sine needs to be calculated.
    - Return Value: The sine of the input angle.
- double Cos(double x):
    - Description: This function returns the cosine of a double in radians.
    - Parameters:
        - x: The angle in radians for which cosine needs to be calculated.
    - Return Value: The cosine of the input angle.
- double Tan(double x):
    - Description: This function returns the tangent of a double in radians.
    - Parameters:
        - x: The angle in radians for which tangent needs to be calculated.
    - Return Value: The tangent of the input angle.
- double operator_add(double lhs, double rhs):
    - Description: This function computes the sum of two double values.
    - Parameters:
        - lhs: The left-hand side operand.
        - rhs: The right-hand side operand.
    - Return Value: The sum of lhs and rhs.
- double operator_sub(double lhs, double rhs):
    - Description: This function computes the difference between two double values (lhs - rhs).
    - Parameters:
        - lhs: The left-hand side operand.
        - rhs: The right-hand side operand.
    - Return Value: The difference between lhs and rhs.
- double operator_mul(double lhs, double rhs):
    - Description: This function computes the product of two double values.
    - Parameters:
        - lhs: The left-hand side operand.
        - rhs: The right-hand side operand.
    - Return Value: The product of lhs and rhs.
- double operator_div(double lhs, double rhs):
    - Description: This function computes the division of two double values (lhs / rhs).
    - Parameters:
        - lhs: The dividend.
        - rhs: The divisor.

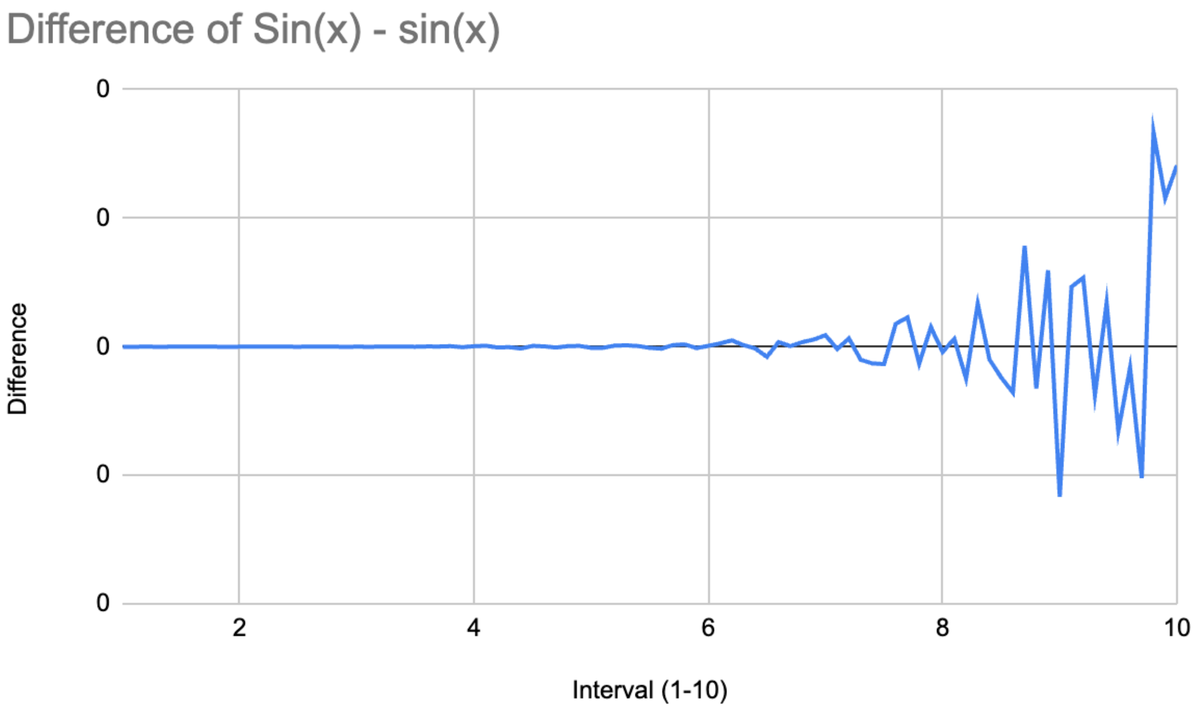
    - Return Value: The result of dividing lhs by rhs.
- bool parse_double(const char *s, double *d):
    - Description: This function attempts to parse a double-precision floating-point number from the input string s. If successful, it stores the parsed number in the memory location pointed to by d and returns true. If the string is not a valid number, it returns false.
    - Parameters:
        - s: The string containing the number to be parsed.
        - d: A pointer to the memory location where the parsed double value will be stored.
    - Return Value: True if the parsing is successful, false otherwise.
-

Results:

```
for (double i = 1; i < 10; i += 0.1) {
      printf("%f", i);
      printf("%.20f\n", Sin(i) - sin(i));
      printf("%.20f\n", Cos(i) - cos(i));
      printf("%.20f\n", Tan(i) - tan(i));
  }
```




Difference of Tan(x) - tan(x)
0
0
0
0
0
0
Difference
2
4
6
8
10
Interval (1-10)


Difference of Sin(x) - sin(x)
0
0
0
0
0
Difference
2
4
6
8
10
Interval (1-10)