Assignment 5 – Calc Report

Max Ratcliff

CSE 13S – Spring 24

Purpose

The purpose of this program is to implement a reverse polish notation calculator that can compute trig functions as well as both bunary and unary operators

Questions

Please answer the following questions before you start coding. They will help guide you through the assignment. To make the grader's life easier, please do not remove the questions, and simply put your answers below the text of each question.

- Are there any cases where our sin or cosine formulae can't be used? How can we avoid this? **if we required super precise output for any reason then our sin and cosine formulae couldnt be used cause its only accurate to a point. in that case we'd either need to shirnk epsilon to make the result more accurate, or use the -m flag to use the provided math.h sin and cos functions which have a much higher degree of accuracy**
- What ways (other than changing epsilon) can we use to make our results more accurate? **we can use trig identies such as** $Tan(x) = \frac{Sin(x)}{Cos(x)}$ **or** $Sin^2(x) + Cos^2(x) = 1$ **this way we only have to compute one of the trig functions and can convert them into others to avoid losing accuracy.** [1]
- What does it mean to normalize input? What would happen if you didn't? **normalizing the input means to make sure the input can be understood by our program by ensuring it looks a certain way or sits between certain values. For this program we will normalize our input for trig functions to be in the range of 0 to** 2π **if we didnt normalize it we would most likely end up with incorrect input**
- How would you handle the expression 321+? What should the output be? How can we make this a more understandable RPN expression? **for since the calculator splits on space it would first push** 321 **to the stack and then treat '+' as a unary operator which would then just output** 321
- Does RPN need parenthesis? Why or why not? **RPN does not need parenthesis because the order of operations is handled by the stack and the calculator just operates down until there are no items remaining in the stack.**

Testing

I will test a variety of eraneuous inputs, such as letters and special characters, I will test to ensure unary operators are handled as well as test to make sure an improper amount of operators are handled. I will also test the bounds of the calculator by providing very very big and very very small numbers

[1] hint: Use trig identities

How to Use the Program

first compile the program with `make` and then run the program with `./calc` with the optional flags `-h` `-m` which brings up the help menu and makes the calculator use trig functions from math.h respectively. You will then be able to provide inputs such as `3 2 1 + +` and recieve output, which in the example case should be 6. You can exit at any time using `ctrl+d`.

Program Design

Most of the functions will be in external files that will be included in the calc.c file. stack.c implements a basic stack with push, pop, peek, print and clear functionality, operators.c contains functions to be turned into function pointers as well as functions to apply the operations and a function to parse a number out of a string. mathlib.c contains sine, cosine, sqrt, and absolute value functions all of which exept the last use either tailor series or the babylonian method

Pseudocode

first process flags, if help flag print help message start the processing of the program and get an expression from stdin into a buffer, then split the expression on whitespace go through the expression token by token, if its a number push it to stack if its a binary operator perform the operation if its a unary operator check if -m flag is present if so use math.h trig fucntions otherwise use the ones written in math.c if its neither a binary or unary operator print error message if there is no error print the stack clear the stack and get the next expression from stdin.

Function Descriptions

For each function in your program, you will need to explain your thought process. This means doing the following

bool apply_binary_operator()

- **Inputs: function pointer**
- **Outputs: true/false as a sucess state**
- **pops two values from the stack and operates on them depending on which function pointer is passed, then pushes result back onto stack used for addition, subtraction, multiplication and division.**

bool apply_unary_operator()

- **Inputs: function pointer**
- **Outputs: true/false as a sucess state**
- **pops a value from the stack and operates on it depending on which function pointer is passed, then pushes result back onto stack used for unary '-' trig functions and sqrt**

char parse_double()

- **Inputs: string and a pointer to a double**
- **Outputs: succes state (true/false), and updated value of the double pointer**
- **uses strtod to attempt to parse a double from a string and give the pointer that value**

bool stack_push()

- **Inputs: double to push to stack**
- **Outputs: success state(true/false)**
- **pushes to the top of the stack ensuring stack size is updated and doesnt exeed stack capacity**

bool stack_peek()

- **Inputs: double pointer**
- **Outputs: success state(true/false), updated pointer value**
- **sets the pointer to be the value on the top of the stack, throws an error if the stack is empty**

bool stack_pop()

- **Inputs: double pointer**
- **Outputs: success state(true/false), updated pointer value**
- **sets the pointer to the value at the top of the stack and decrement the stack size, returns false if stack is empty**

bool Abs()

- **Inputs: double**
- **Output: double**
- **if the double is positive return it otherwise return its opposite**

bool Sqrt()

- **Inputs: double**
- **Outputs: double**
- **uses the baylonian method to find and return the square root of the double**

bool Sin()

- **Inputs: double**
- **Outputs: double**
- **normalizes the input to be between $0 - 2\pi$ and then uses the mclauren series for sin to approximate it stoping when the next term is EPSILON which is a value set in `mathlib.h` to 10^{-14}.**

bool Cos()

- **Inputs: double**

- **Outputs: double**
- **normalizes the input to be between $0 - 2\pi$ and then uses the mclauren series for cosine to approximate it stoping when the next term is EPSILON which is a value set in `mathlib.h` to 10^{-14}.**

bool Tan()

- **Inputs: double**
- **Outputs: double**
- **normalizes the input to be bewtween $0 - 2\pi$ and then uses the trig identity: $\tan x = \frac{\sin x}{\cos x}$ to approximate tangent**

Results

Follow the instructions on the pdf to do this. In overleaf, you can drag an image straight into your source code to upload it. You can also look at `https://www.overleaf.com/learn/latex/Inserting_Images`