

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 binary 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 couldn't be used cause its only accurate to a point. in that case we'd either need to shrink 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 identities 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 didn't 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 erroneous 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

¹hint: 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 receive 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 except the last use either Taylor 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 functions 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

- The inputs of every function (even if it's not a parameter)
- The outputs of every function (even if it's not the return value)
- The purpose of each function, a brief description about a sentence long.
- For more complicated functions, include pseudocode that describes how the function works
- For more complicated functions, also include a description of your decision making process; why you chose to use any data structures or control flows that you did.

Do not simply use your code to describe this. This section should be readable to a person with little to no code knowledge. **DO NOT JUST PUT THE FUNCTION SIGNATURES HERE. MORE EXPLANATION IS REQUIRED.**

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