Assignment 5 – Calc Report Template

Mira Saini

CSE 13S – Spring 24

Purpose

Audience for this section: Pretend that you are working in industry, and write this paragraph for your boss. You are answering the basic question, "What does this thing do?". This section can be short. A single paragraph is okay.

Do not just copy the assignment PDF to complete this section, use your own words.

This program will act as a scientific calculator, performing computations like addition, subtraction, multiplication, division, sine, cosine, tangent, absolute value, and square root. The program will implement a stack ADT to push and pop values entered by the user in Reverse Polish notation. The program will then output the calculated expression.

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?
  this might not be right
  When 'n' in the sin/cos formulas is equal to 0, we're dividing by 0 which isn't allowed. We can avoid this though by making sure we are doing float division rather than integer division because float will give an approximation and never actually equal 0.

- What ways (other than changing epsilon) can we use to make our results more accurate? [1]

  Using the Taylor series, we can increase the degree of the polynomial to make the results more accurate/change epsilon in addition to increasing the precision of the floating point numbers in the program to reduce rounding errors.

- What does it mean to normalize input? What would happen if you didn't?

  To normalize input, specifically for this assignment, means to make sure that the user input is converted to a radian angle between 0 and 2pi. Not normalizing the input can lead to inconsistencies and inaccuracies when the program calculates the expression for values outside the range 0 to 2pi.

- How would you handle the expression 321+? What should the output be? How can we make this a more understandable RPN expression?

  Since there isn't two numbers, the function bool apply_binary_operator() would return false and error handling would be done in calc.c to print the error message "error: not enough values on stack for binary operator". To make this an understandable RPN expression we could do '321 0 +' instead, which would perform the same assuming that we just want the result to be '321.00000'.

[1] hint: Use trig identities

- Does RPN need parenthesis? Why or why not?

  No, RPN does not need parenthesis because the order of operators and numbers makes it explicitly known which computations are performed first.

Testing

List what you will do to test your code. Make sure this is comprehensive. [2] Be sure to test inputs with delays and a wide range of files/characters.

Testing RPN Network and Calculator:
1. 3 4 +
2. 9 + must have two args
3. 2 9 -
4. 0 4 *
5. 9 0 / should print 'inf' because cannot divide by 0
6. 3 5 /
7. 5 3 + 9 z should print error message

Testing Math and Getopt: Using getopt to test trig functions from mathlib and my own trig functions
Making sure that ./calc -h prints the usage message
Testing Error Handling: Making sure that if the function apply_binary_operator fn and apply_unary_operator fn functions return false, then there respective error messages from messages.h is printed
Making sure that if the stack is at capacity, then the error message that states there isn't enough space is printed
Making sure that if one of the chars in the input is not one of the identified chars, then an error message ERROR_BAD_CHAR prints.

I should first make my tests directory in my asgn5 directory using mkdir. Inside there, I can create test scripts that use diff to compare the output of the executable binary and my own program. I can test different inputs such as the ones above. I should also create a runner.sh file to make it easier to run all of my tests at once. Once I've created this file, I can add a target to my Makefile as well to run my tests.

Example test script: 1. Ensuring that my executables are in my test folder 2. My delayinput.sh is in my test folder 3. Make target for tests and bash runner.sh 4. runner.sh in asgn5 dir 5. create test : vim test_4+.sh //this will test the input 4 +/n 6. test script could look something like this:

```
./calc_x86 4 + > expected.txt
./calc 4 + > actual.txt


if diff expected.txt actual.txt > /dev/null; then
    echo "Your file did not output the expected"
    rm actual.txt
    rm expected.txt
    exit 1
fi


echo "Test was a success, the program produces the right error!"

rm actual.txt
rm expected.txt
exit 0
```

Run 'make' and then bash runner.sh to run all tests.

[2]This question is a whole lot more vague than it has been the last few assignments. Continue to answer it with the same level of detail and thought.

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 this program, start the calculator by doing ./calc. You may choose to include a ’-h’ or a ’-m’ in your command line. If you do a ’-h,’ a usage message will appear. ’-m’ will use the built-in trig functions instead of the functions written in the program that approximate the functions instead.

Then give your arguments (values and operands) after the ’¿’ in Reverse Polish Notation where two operands and given first and then the operator last. For example, ”3 4 +” is valid RPN. The program will compute and output the result and then a ’¿’ will appear again to signal that additional inputs can be given.

To show “code font” text within a paragraph, you can use `\lstinline{}`, which will look like this: `text`.

For a code block, use `\begin{lstlisting}` and `\end{lstlisting}`, which will look like this:

```
Here is some code in lstlisting.
```

And if you want a box around the code text, then use `\begin{lstlisting}[frame=single]` and `\end{ lstlisting}`
which will look like this:

```
Here is some framed code (lstlisting) text.
```

Want to make a footnote? Here’s how.[3]

Do you need to cite a reference? You do that by putting the reference in the file `bibtex.bib`, and then you cite your reference like this[1][2][3].

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.

My program has multiple files that contain different functions for my stack, binary operators, and unary operators, and their corresponding header files. I have a messages.h file which contains the error/usage messages. In my main function file, I first list all of my header files and then define my buffer size. Then I declare two global variables that will be used in my getopt portion of my program. Inside my main function, I declare a string array and initalize it to the name of my program. I then use getopt to define the ’h’ and ’m’ tags and use if statements to either print the usage statement for the former or use the standard library math trig functions for the latter. After defining getopt, I use a while(1) loop that loops through the rest of my program, taking in user input through stdin until the user ends the program using ’Ctrl-C.’ I print ”¿ ” to stderr and then read the user input into a buffer. I then parse the buffer using a space as a delimiter, going through each token char in the buffer. Inside another while loop, which exit if error is set to true or the token equals null, I have multiple if and if-else statements that determine what type of char it is (’+’ ’-’, ’4’,’s’) and then executes actions (either pushing to the stack if its a number, or calling a function to apply the operator, or printing an error message). Once an error message pops up or the end of the input is reached, the stack prints as long as there wasn’t an error message that ended the while loop. Then the whole stack is cleared and then the user is prompted for more input.

[3]This is my footnote.

Pseudocode

Give the reader a top down description of your code! How will you break it down? What features will your code have? How will you implement each function.

FUNCTIONS: stack.c:

```
//initalize stack_size to 0
//declare stack

bool stack_push(double item){
    if stack_size < stack_capacity{
        stack[stack_size] = item;
        //increment stack_size
        return true;
    }else{
        return false;
    }
}

bool stack_peek(double *item){
    if stack_size > 0{
        //set item equal to the last item in the stack
        return true;
    }else{return false;}
}

void stack_clear(void){
    stack_size = 0;
}

void stack_print(void){
    if stack_size == 0 {return;}
    //print stack using for loop
}
```

operators.c

```
//include header files

bool apply_binary_operator(binary_operator_fn op){
    if stack_size < 2 {return false}
    double y;
    assert(stack_pop(&y))
    //do the same for x
    double result
    //initialize result to op(x,y) and assert that push //occured
    return true
}

bool apply_unary_operator(unary_operator_fn op){
    if stack_size < 1{return false}
    double x
    assert(stack_pop(&x))
    double result = op(x)
    //assert result is pushed
    return true
}

double operator_add(double lhs, double rhs){
    //return lhs + rhs
}
```

```
//do basically the same simple operations for sub, mul, div
.
.
.
bool parse_double(const char *s, double *d){
    char *endptr
    double result = strtod(s, &endptr)
    if endptr != s{
        //set address at pointer d equal to result
        return true
    }
    else{return false}
}
```

mathlib.c

```
//define Epsilon and include header files

double Abs(double x){
    if x<0{
        x = -x
    }
    return x
}

double Sqrt(double x){
    if x<0{
    return nan("nan")
    //declare and initialize old and new
    //while loop that checks that the abs value of old- new is > EPSILON
    //reassign old to new and update new
    }
    return new
}

double Sin(double x){
    //normalize the input using fmod
    if x > 2pi{
        x = 2pi - x
    }

    if x < 0{
        x = 2pi + x
    }
    //intialize sum = 0, x_to_the_k = x, k_fact = 1.0, k = 1.0, neg = 1.0
    while(1){
        double term = x_to_the_k * neg / k_fact
        if Abs(term) < EPSILON{break}
        //add term to sum total, increment x_to_the_k, k_fact, k, neg
    }
    return sum
}


double Cos(double x){
    //normalize the input
    if x > 2pi{
        x = 2pi - (x - 2pi)
    }
    if x < 0{
        x = 2pi + x
    }
    //intialize sum = 0, x_to_the_k = 1.0, k_fact = 1.0, k = 0.0, neg = 1.0
    while(1){
        double term = x_to_the_k * neg / k_fact
        if Abs(term) < EPSILON{break}
        //add term to sum total, increment x_to_the_k, k_fact, k, neg
    }
    return sum
}

double Tan(double x){
    double tan = Sin(x) / Cos(x)
    return tan;
}
```

MAIN FUNCTION FILE:

```
//include all my header files
//define BUFFER_SIZE 1024
//initalize global vars "breaks" and mathlib"
int main(int argc, char **argv0{
    int opt
    char program[7]
    //use strncpy to copy argv[0] into program string
    while((opt = getopt(argc, argv, "hm")) != -1){
        if opt == 'h'{
            //print usage message
            breaks = 1
            break
        }else if opt == 'm'{
            mathlib = 2
        }
    }
    if breaks == -1{exit(0)}
    while(1){
        //print "> "
        //declare and initalize buffer
        //use fgets to read stdin input into buffer
        //get rid of newline at the end of buffer

        char *saveptr
        bool error = false
        const char *token = strtok_r(buffer, " ", &saveptr)
        double x
        while(token != NULL and !error){
            //use parse_double function and check if token could be pushed onto stack using
                stack_push, if not print error and set error to true
            else if(strcmp(token, "+") == 0){
                if apply_binary_operator(operator_add) != true{
                //printf error message
                //set error to true
                }
            }
            .
            .
            . // do this for all binary and unary operators
            //for unary operators, check if mathlib == 2, if so then use the standard lib math
                functions
            .
            .
            .
            //call the strtok_r function again

            if(!error){
                stack_print()
                //print newline
            }
            stack_clear()
        }







    }
}
```

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.**

Functions from stack.c

bool stack_push(double item): This function has a floating point number parameter which is pushed to the stack. The last item in the stack is reassigned to item if the stack_size is less than the stack_capacity. Then the stack_size is incremented and the function returns true if successful, otherwise false.

bool stack_peek(double *item): This function takes in a pointer to a floating point number as a parameter. If the stack_size is greater than 0, then the pointer to the address is assigned the last item in the stack and then returned true, otherwise it returns false.

bool stack_pop(double *item): This function takes in a pointer to a floating point number as a parameter. If the stack_size is greater than 0, then the pointer to the address is reassigned to the value of the last item in the stack and then the stack_size is decremented by 1 and then the function returns true, otherwise false.

void stack_clear(void): This clear the stack by setting the stack_size to 0.

void stack_print(void): If the stack_size equals 0, then the function just returns, otherwise the function prints the stack using a for loop.

Functions in mathlib.c double Abs(double x): This will return the absolute value of x. The parameter is x.

double Sqrt(double x): This function takes a double floating point number as var x and then returns the square root of x. If x is less than 0, it returns nan("nan"). Otherwise, it uses an algorithm to square x.

double Sin(double x): This function returns the sin of x and take it in as the parameter. It has multiple vars: double sum, x_to_the_k, k_fact, k, and neg. It uses a while loop and assigns the double term to one part of the Taylor series expression and then checks if the absolute value of the term is less than epsilon. If so, then it adds the term to the sum and then increments the vars until the while loop breaks. Then the sum of the terms is returned which is the approximation of the sin(x). I already provided the pseudocode for this function and all other functions in the pseudo code section.

double Cos(double x): This function returns the cosine of x and takes x in as the parameter. It has the same structure as sin as it normalizes the input and uses the exact same vars, except they are initialized to different values. The process of calculating the term and then adding it to the sum is the same as the Sin function. The sum is then returned after the while loop breaks. Pseudocode already provided.

double Tan(double x): This function returns the tangent of x and takes x in as a parameter. It calls two functions, the Cos and Sin function and returns the Sin(x) / Cos(x).
operators.c functions:

bool apply_binary_operator(binary_operator_fn op): This function applies any binary operator like '+,' '-,' '*,' '/,' and 'fmod' to the stack values and returns either true if it was a success or false if the stack_size is too small for the binary operator to be applied.

bool apply_unnary_operator(unary_operator fn op): This function does a similar task like the binary function above, except that it checks if the stack_size is at least 1 because the unary operators only need one value to work. If the unary operator is applied, then function returns true, otherwise false.

bool operator_add(double lhs, double rhs): Sums two values together and returns it.

bool operator_sub(double lhs, double rhs): Takes the lhs value and subtracts it by the rhs and returns it.

bool operator_mul(double lhs, double rhs): Returns the product of lhs and rhs.

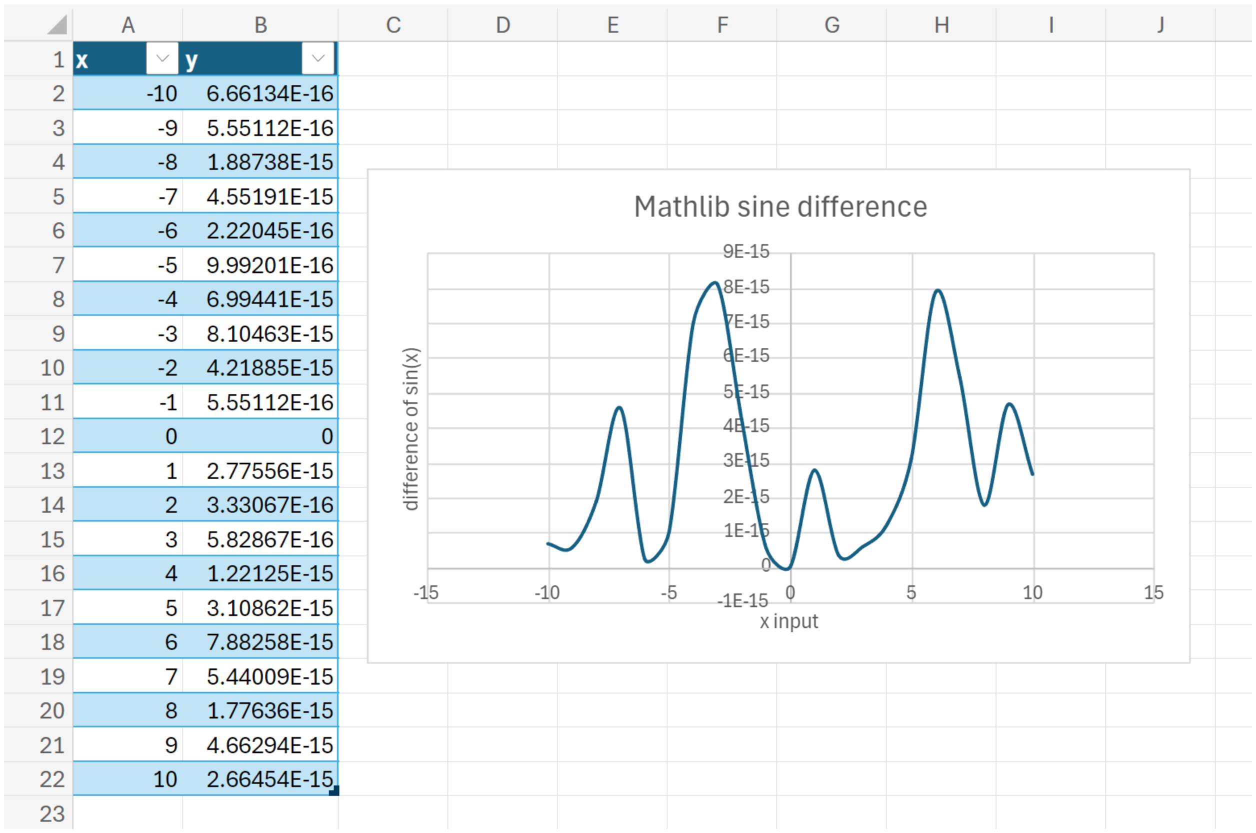
bool operator_div(double lhs, double rhs): Takes lhs and divides it by rhs and returns it.

bool parse_double(const char *s, double *d): Takes a string and a floating point and determines if the floating point is valid. Then returns true if it is, otherwise false.

Results

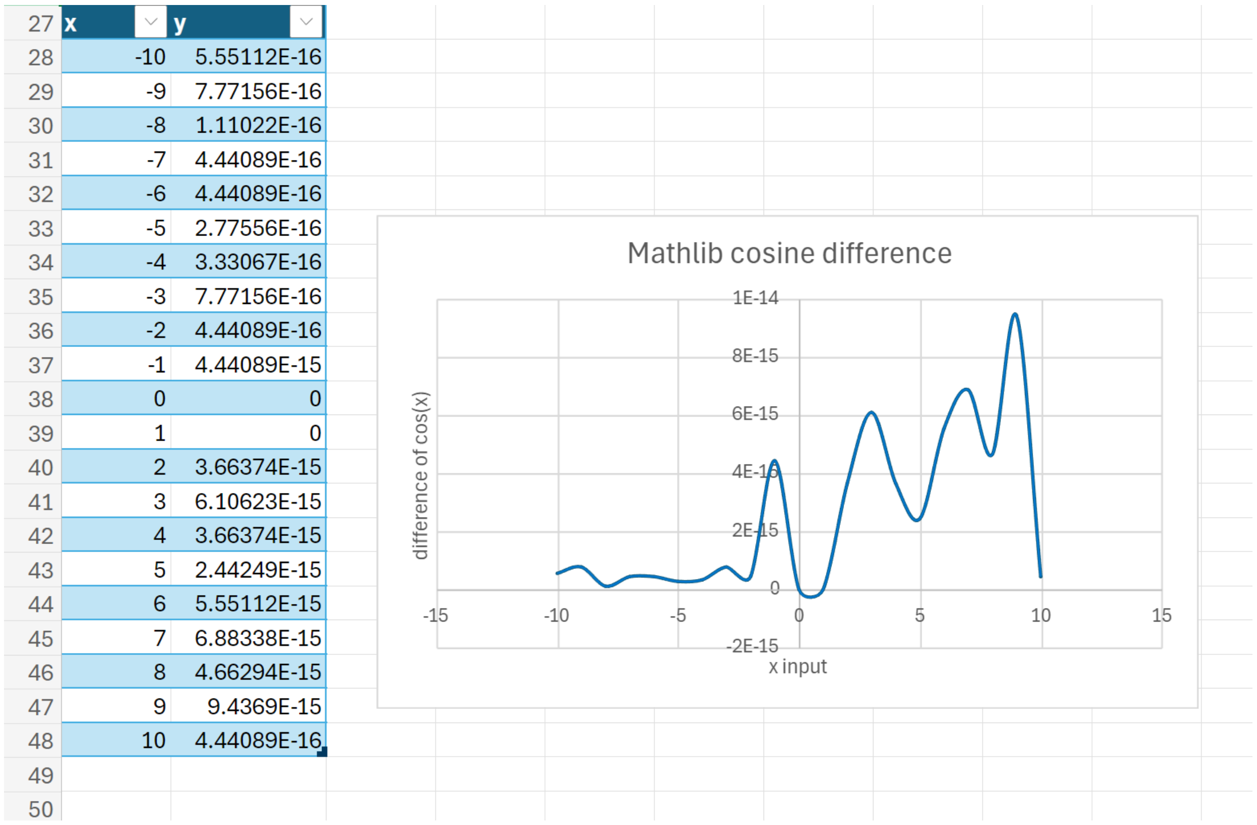
Follow the instructions on the pdf to do this. In overleaf, you can drag an image straight into your source code to upload it. You can also look at `https://www.overleaf.com/learn/latex/Inserting_Images`

To get this data, I created a program that took in the sine, cosine, and tangent values of my functions and calculated the values of each trig function from -10 to 10 and then I subtracted those values for the standard built in trig functions from -10 to 10. Then I took the absolute value of those values and printed them to a file. I then had a file full of the differences between my functions and the built in functions from



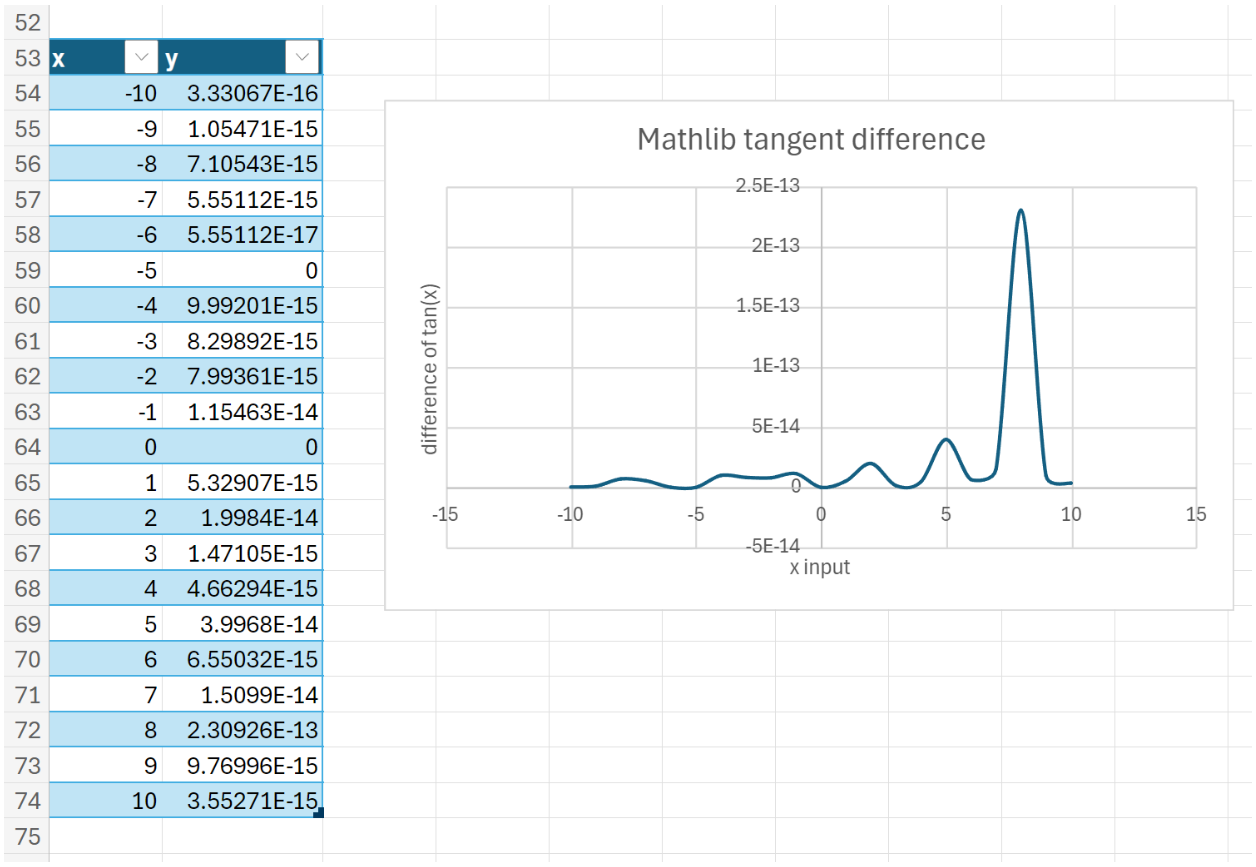
x	y
-10	6.66134E-16
-9	5.55112E-16
-8	1.88738E-15
-7	4.55191E-15
-6	2.22045E-16
-5	9.99201E-16
-4	6.99441E-15
-3	8.10463E-15
-2	4.21885E-15
-1	5.55112E-16
0	0
1	2.77556E-15
2	3.33067E-16
3	5.82867E-16
4	1.22125E-15
5	3.10862E-15
6	7.88258E-15
7	5.44009E-15
8	1.77636E-15
9	4.66294E-15
10	2.66454E-15

Figure 1: Sine difference graph



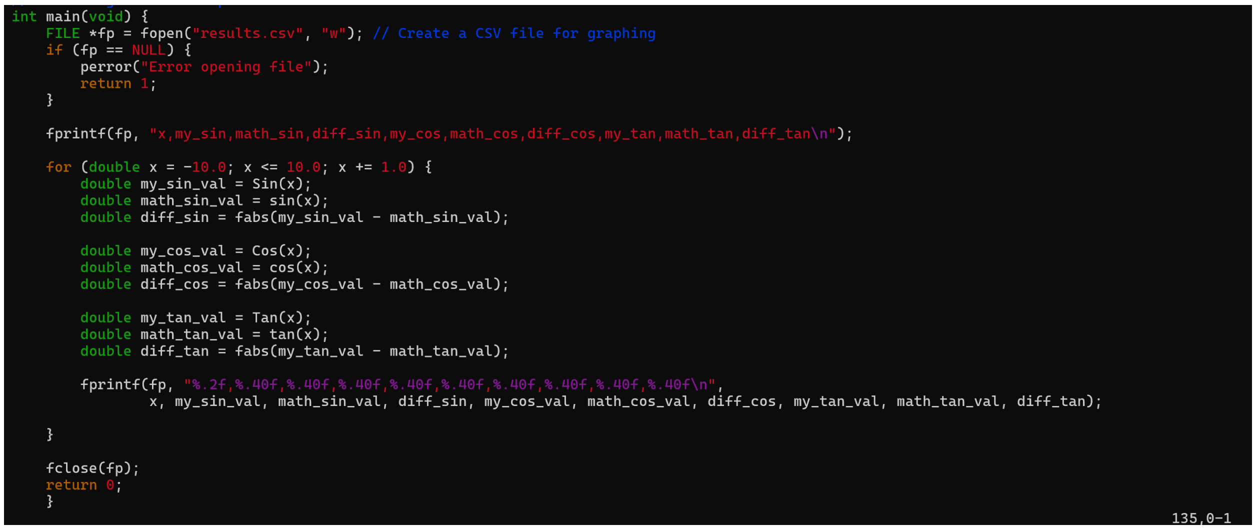
x	y
-10	5.55112E-16
-9	7.77156E-16
-8	1.11022E-16
-7	4.44089E-16
-6	4.44089E-16
-5	2.77556E-16
-4	3.33067E-16
-3	7.77156E-16
-2	4.44089E-16
-1	4.44089E-15
0	0
1	0
2	3.66374E-15
3	6.10623E-15
4	3.66374E-15
5	2.44249E-15
6	5.55112E-15
7	6.88338E-15
8	4.66294E-15
9	9.4369E-15
10	4.44089E-16

Figure 2: Cosine difference graph



x	y
-10	3.33067E-16
-9	1.05471E-15
-8	7.10543E-15
-7	5.55112E-15
-6	5.55112E-17
-5	0
-4	9.99201E-15
-3	8.29892E-15
-2	7.99361E-15
-1	1.15463E-14
0	0
1	5.32907E-15
2	1.9984E-14
3	1.47105E-15
4	4.66294E-15
5	3.9968E-14
6	6.55032E-15
7	1.5099E-14
8	2.30926E-13
9	9.76996E-15
10	3.55271E-15

Figure 3: Tangent difference graph



```
int main(void) {
    FILE *fp = fopen("results.csv", "w"); // Create a CSV file for graphing
    if (fp == NULL) {
        perror("Error opening file");
        return 1;
    }

    fprintf(fp, "x,my_sin,math_sin,diff_sin,my_cos,math_cos,diff_cos,my_tan,math_tan,diff_tan\n");

    for (double x = -10.0; x <= 10.0; x += 1.0) {
        double my_sin_val = Sin(x);
        double math_sin_val = sin(x);
        double diff_sin = fabs(my_sin_val - math_sin_val);

        double my_cos_val = Cos(x);
        double math_cos_val = cos(x);
        double diff_cos = fabs(my_cos_val - math_cos_val);

        double my_tan_val = Tan(x);
        double math_tan_val = tan(x);
        double diff_tan = fabs(my_tan_val - math_tan_val);

        fprintf(fp, "%.2f,%.40f,%.40f,%.40f,%.40f,%.40f,%.40f,%.40f,%.40f,%.40f\n",
                x, my_sin_val, math_sin_val, diff_sin, my_cos_val, math_cos_val, diff_cos, my_tan_val, math_tan_val, diff_tan);

    }

    fclose(fp);
    return 0;
    }
                                                                                                                135,0-1
```

Figure 4: Snapshot of code

the value -10 to 10. I put these values in a table in Excel and then graphed them.

I will submit this program file as well. Here is a snapshot of the code:

I set the floating point to 40 decimals to get accurate difference values. Since the algorithm for my trig functions is based off the Taylor series which is an approximation summing multiple polynomials, there is bound to be very minimal differences between the actual values and my approximated value. If I wanted to get an even more accurate approximation, I could set EPSILSON to be a very tiny number, which would make my algorithm loop through and add even smaller terms to my sum and become more accurate. The error can be reduced more if more terms are used to approximate the value.

References

[1] Wikipedia contributors. C (programming language) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/C_(programming_language), 2023. [Online; accessed 20-April-2023].

[2] Robert Mecklenburg. *Managing Projects with GNU Make, 3rd ed.* O'Reilly, Cambridge, Mass., 2005.

[3] Walter R. Tschinkel. Just scoring points. *The Chronicle of Higher Education*, 53(32):B13, 2007.