Diwa Ashwini Vittala
CSE 13S - Fall 2021

⁰❀ Assignment 2: Design ❀。

A LITTLE SLICE OF π

About the program:

The program can print the square root of values from 0-10 incremented by 0.1, print the value of e and print the value of pi in comparison to the values in the math library, math.h.
What each function does:
e(): approximate the value of e as a double
e_terms(): returns the number of iterations it took for e() to calculate the value
pi_euler(): approximates the value of pi as a double using the Euler method
pi_euler_terms(): returns the number of iterations it took for pi_euler() to calculate the value
pi_bbp(): approximates the value of pi as a double using the Bailey Borwein Plouffe
pi_bbp_terms(): returns the number of iterations it took for pi_bbp() to calculate the value
pi_madhava(): approximates the value of pi as a double using the Madhava Series
pi_madhava_terms(): returns the number of iterations it took for pi_madhava() to calculate the value
pi_viete(): approximates the value of pi as a double using the Viete formula
pi_viete_factors(): returns the number of iterations it took for pi_viete() to calculate the value
sqrt_newton(): calculates the approximate value of the square root of a given number and returns it as a double using Newton method
sqrt_newton_iters(): returns the number of iterations it took for sqrt_newton() to calculate the value

About the file mathlib-test.c:

Input(s): in line command of a, e, b, m, r, v, n, s, and h are given during execution (for example, ./mathlib-test.c -e). Multiple commands may be given.
Output: the program outputs a comparison of the related value with the value in the math library.

Other files involved:

Makefile, mathlib-test.h, README.md, e.c, madhava.c, euler.c, bbp.c, viete.c, newton.c, WRITEUP.pdf

Pseudocode/solution breakdown for:

WRITEUP.pdf

Includes analysis about the functions and includes graphs that compares the functions in the math library with the functions in our library.

mathlib.h

Make sure that this file is mentioned only once using #pragma once

This file is provided for us.
We define EPSILON as the number 1e-14
We create a function to find the absolute value of a function by checking if the input
    is less than 0. If it's smaller, find the negative of that number.
    Otherwise, return itself because it must be positive
Declare all the functions we are using, like e, euler, bbp, madhava, viete, newton,
    and their term counts

mathlib-test.c:

Include all necessary files (mathlib, standard input/output, library, boolean, math, getopt, etc)
Begin main using getopt format so that the function takes characters of input:
    Let an integer variable for chosen option be 0
    Set a boolean variable false for every possible option from a through h
    If the number of given arguments from the inline command is 1:
        Set help boolean to true so that a message about correct use shows up
    While loop that begins options for getopt, with possibilities a,e,b,m,r,v,n,s, and h:
        Begin a switch that looks at the chosen option:
        In the case that the user has chosen "a":
            Set the respective boolean to true
            Break out of this switch
        In the case that the user has chosen "e":
            Set the respective boolean to true
            Break out of this switch
        In the case that the user has chosen "b":
            Set the respective boolean to true
            Break out of this switch
        In the case that the user has chosen "m":
            Set the respective boolean to true
            Break out of this switch
        In the case that the user has chosen "r":
            Set the respective boolean to true
            Break out of this switch
        In the case that the user has chosen "v":
            Set the respective boolean to true
            Break out of this switch
        In the case that the user has chosen "n":
            Set the respective boolean to true
            Break out of this switch
        In the case that the user has chosen "s":
            Set the respective boolean to true

Break out of this switch
In the case that the user has chosen “h”:
Set the respective boolean to true
Break out of this switch
If none of the selections were true (such as a case where user types ./mathlib-test -):
Set help to true
If the boolean for “a” selection was true:
Set all other options to true except statistics and help
If e was selected:
Print out the value of e from the e function,
real e, and the difference between them
If user selected for statistics as well (s was selected):
Print out the number of terms used in our e function
If r was selected:
Print out the value of pi from our euler function,
real pi, and the difference between them
If user selected for statistics as well (s was selected):
Print out the number of terms used in our euler function
If b was selected:
Print out the value of pi from our bbp function,
real pi, and the difference between them
If user selected for statistics as well (s was selected):
Print out the number of terms used in our bbp function
If m was selected:
Print out the value of pi from our madhava function,
real pi, and the difference between them
If user selected for statistics as well (s was selected):
Print out the number of terms used in our madhava function
If v was selected:
Print out the value of pi from our viete function,
real pi, and the difference between them
If user selected for statistics as well (s was selected):
Print out the number of terms used in our viete function
If n was selected:
For every number between 1 and 10 incremented by 0.1:
Print the value of its square root from our newton function,
the real square root using the math library,
and the difference between them
If user selected for statistics as well (s was selected):
Print out the number of terms used in our square root function

for each number
If the user only selected s but nothing else:
Set help as true
If help is true, h was selected, or if the user did not specify any particular command:
Print out instructions on the proper usage of mathlib-test.c file

e.c:

Include the mathlib.h file
Declare a static integer to keep track of what term we are on

Begin the e() function:
If the variable counting the terms is not 0, them set it equal to 0
Declare a variable for sum (which is 0 at this point) and
fraction (which is 1 right now since we are considering
the 0th iteration of this function before going into the loop)
Start a while loop to keep going until our fraction is smaller than EPSILON:
If we are on the first term:
Make sure fraction is 1 and add it to sum
Otherwise:
Divide the fraction by the term number and set that as the new fraction
Add the fraction to the sum
Increment the count by 1 (addition)
Update the sum by adding in the fraction
Return the sum

Begin a function, e_terms(), to keep track of the number of terms:
Return the static variable we declared at the top

madhava.c:

Include the mathlib.h file
Declare a static integer to keep track of what term we are on

Begin the madhava function:
If the variable counting the terms is not 0, them set it equal to 0
Declare variables to keep track of the sum, fraction, exponent part
and 2k+1 part of the fraction
Start a while loop to keep going until our fraction is smaller than EPSILON:
If we are on the first term:
Make sure fraction is 1
Otherwise:

Calculate the exponent part of the fraction by multiplying itself by -3
Calculate the second term in the bottom of the fraction by
multiplying the count by 2 and adding 1
Set fraction as top divided by bottom and then add it to the sum
Update the sum by adding in the fraction
Increment the count by 1 (addition)
Return the product of the sum and the square root of 12 using the newton function

Begin a function, pi_madhava_terms(), to keep track of the number of terms:
Return the static variable we declared at the top

euler.c:
Include the mathlib.h file
Declare a static integer to keep track of what term we are on

Begin the euler function:
If the variable counting the terms is not 0, them set it equal to 0
Declare variables to keep track of the sum, fraction, and term count
For a variable, loop until fraction is smaller than epsilon:
If we are on the first term:
Make sure fraction is 1
Otherwise:
Make the fraction 1/(the term count)^2
Add fraction to the sum and increment the iteration counts
Return the square root of 6 times the sum

Begin a function, pi_euler_terms(), to keep track of the number of terms:
Return the static variable we declared at the top

bbp.c:
Include the mathlib.h file
Declare a static integer to keep track of what term we are on

Begin the bbp function:
If the variable counting the terms is not 0, them set it equal to 0
Declare variables to keep track of the sum, fraction, parts of the fraction,
the 16^-k part, and entire term
Start a while loop to keep going until our entire term is smaller than EPSILON:
Calculate the top of the fraction in the Horner normal form
Calculate the bottom of the fraction in the Horner normal form

Set fraction as top/bottom
Let's start calculating the 16^-k term with if we are on the first term:
Set the 16^-k part to 1
Otherwise,
Set the 16^-k part as itself multiplied by (1/16)
Increment the term count by 1 (addition)
Set the whole term equal to the 16^-k term multiplied by the fraction
Add the term to the sum
Return the sum

Begin a function, pi_bbp_terms(), to keep track of the number of terms:
Return the static variable we declared at the top

viete.c:
Include the mathlib.h file
Declare a static integer to keep track of what term we are on

Begin the viete function:
Make sure we are starting with a count of 0 iterations
Declare a variable as square root of 2
Declare a variable to keep track of the product, fraction, and the top of the fraction
While the absolute value of the difference between 2 and the top of our
fraction is smaller than EPSILON:
If we are on our first iteration:
Make sure top is square root of 2
Otherwise, make top = square root of 2 + the previous top
Assemble the fraction (so that fraction is top divided by 2)
Update the product by multiplying the new fraction into it
Increment the term count
Return the value of 2 divided by the product

Begin a function, pi_viete_factors(), to keep track of the number of terms:
Return the static variable we declared at the top

newton.c:
Include the mathlib.h file
Declare a static integer to keep track of what term we are on

Begin the square root function, which takes in a double-type number as input:
Make sure we are starting from a count of 0.

Declare two variables, y and z, as doubles
While the absolute value of the difference between y and z is smaller than EPSILON:
Let z be the same number as y so we can find the middle
Set y as half of (the input number/z) plus z
Increment the iteration count
Return the value of y

Begin a function, sqrt_newton_iters(), to keep track of the number of terms:
Return the static variable we declared at the top