Assignment 2
Design
Prof Veenstra
Nguyen Vu

1. e()

I create a temporary denominator, numerator and e variable that starts at 0. After that I make a loop so that it keeps adding the new right terms using Taylor series to the e variable by reusing the denominator from previous term until the new term is smaller than EPSILON.

Pseudo Code:
denominator = 0
e = 0
term = 1
for loop
term = 1/denominator
make the new base for next term
add term to e

1.2. e_terms()

I create a loop count as the global variable and add 1 to it every time it loops in e()

Pseudo Code:
global loop = 0
for every loop in e(), add 1 to global loop

2.1. pi_madhava()

I create a temporary denominator, numerator

$$(-3)^{-}k$$

and pi variable that starts at 0. After that I make a loop so that it keeps adding the new right terms using the Madhava's method to the pi variable by computing the new denominator multiply by the new

$$(-3)^{-}k$$

value. After the new term is smaller than EPSILON, I multiply the final value by

$$\sqrt{12}$$

Pseudo Code:
term = 1

pi = 0
numerator = 1
for loop
term = numerator / (2k + 1)
add term to pi
compute new numerator
compute

$$\sqrt{12}$$

and multiply it to the final pi value

2.2. pi_madhava_terms()

I create a loop count as the global variable and add 1 to it every time it loops in pi_madhava()
Pseudo Code:
global loop = 0
for every loop in pi_madhava(), add 1 to global loop

3.1. pi_euler()

I create a pi variable. After that I make a loop so that it keeps adding the new right terms using the Euler's method. After the new term is smaller than EPSILON, I multiply the pi variable by 6 then square root the result for the final pi value.
Pseudo Code:
pi = 0
for loop:
term =

$$1/(k^2)$$

add 1 to k
add term to pi
multiply pi by 6
square root the result

3.2. pi_euler_terms()

I create a loop count as the global variable and add 1 to it every time it loops in pi_euler()
Pseudo Code:
global loop = 0
for every loop in pi_euler(), add 1 to global loop

4.1. pi_bbp()

I create a pi variable. After that I make a loop so that it keeps adding the new right terms using the Bailey-Borwein-Plouffe's method with k increases

by 1 every loop until the new term is smaller than EPSILON.
Pseudo Code:
pi = 0
base = 1
for loop:
term = (1.0 / base) * ((4.0 / ((8.0 * k) + 1.0)) - (2.0 / ((8.0 * k) + 4.0)) - (1.0 / ((8.0 * k) + 5.0))- (1.0 / ((8.0 * k) + 6.0)))
multiply base by 16
add term to pi

4.2. pi_bbp_terms()

I create a loop count as the global variable and add 1 to it every time it loops in pi_bbp()
Pseudo Code:
global loop = 0
for every loop in pi_bbp(), add 1 to global loop

5.1. pi_viete()

First I find the square root of 2 then divide it by 2 and use it as the first term. After that I make a for loop and multiply the previous term by 2, add 2 then square root the result. Then I take the result and divide it by 2 and finally multiply is with the previous result. I keep repeating that process until the new result subtract the previous one is smaller than EPSILON.
Pseudo Code: term =

$$\sqrt{2}/2$$

add term to pi for loop:
new term =

$$\sqrt{2 + (term * 2)}$$

multiply pi by new term
update current term by multiplying with new term

5.2. pi_viete_factors()

I create a loop count as the global variable and add 1 to it every time it loops in pi_viete()
Pseudo Code:
global loop = 1 because the first term is

$$\sqrt{2}/2$$

for every loop in pi_viete(), add 1 to global loop

6.1. sqrt_newton()
I create an initial guess for the square root of the given value. After that I keep calculating new square root value using the Newton-Raphson method until the new guess subtract the old one is less than EPSILON.
Pseudo Code:
guess = 1
for loop:
function =

$$guess^2 - givenvalue$$

derived function = 2*guess
guess = guess = function/(derived function)
6.2. sqrt_newton_iters()
I create a loop count as the global variable and add 1 to it every time it loops in sqrt_newton()
Pseudo Code:
global loop = 0
for every loop in sqrt_newton(), add 1 to global loop
7. mathlib-test.c
I create every test for respective letter of operation similar to the example and set -s to turn on/off the return terms function and -h to shows the main menu.
Pseudo Code:
opt = 0
return term = 0
while loop for opt: switch (opt)
Print and compare value from respective functions and actual mathematical value with respective command letter
case 's': return term = 1
case 'h': prints the main menu
8. Makefile
I create commands to compiles the mathlib-test.c into executable and other math files into .o files.
Pseudo Code:
CC = clang
CFLAGS = -Wall -Wpedantic -Werror -Wextra
OBJS = e.o madhava.o euler.o bbp.o pi_viete.o newton.o
set all: command to compile mathlib-test
set mathlib-test: command to compile mathlib-test.c and other math files.c

set clean: command to remove all created files
set format: command to clang format any .c files
set %.o: %.c command to compile any .c files into .o files