Assignment 2: A Little Slice of pi

Kelly Liu

October 6, 2021

1 Description

In this program, we are making a bunch of different math functions that is usable with the command prompt. Making an e function that uses a taylor series to approximate e. An madhava function to approximate pi using the Madhava series. An euler function to approximate pi using Euler's formula. An bbp function to approximate pi using the Bailey-Borwein-Plouffe formula. An viete function to approximate pi using Viete's formula. A newton function to approximate the square root of an argument using the Newton-Raphson method. Lastly, an mathlib test file to run our functions and tests.

2 Pseudocode

2.1 e.c

```
import all files
set a static counter variable
set a total amount to 0
set a start variable to 0
set a hold variable to 0
use a while loop to run until it hits smaller than epsilon
set a bottom double variable to 1
use a for loop to loop from 1 to the iteration of start unless start is 0(if start is 0 simply set bottom to 1)
update bottom variable to itself times the iteration of the for loop
end of inner for loop
update the bottom variable to 1 divided by bottom variable
set a if statement to check if values are smaller than epsilon
end of if statement
update total to itself plus temp variable
update hold and start and counter variables
end of outer for loop
return total
```

For the e_terms part, return static variable

2.2 madhava.c

```
import all files
set a static counter variable
set a total, start and hold variables to 0
use a while loop to run until it hits smaller than epsilon
set a temp double variable to -3
use a for loop to loop from 0 to the iteration of start unless start is 0(if start is 0 simply set bottom to 1)
update the temp variable to itself times -3
end of for loop
multiple the temp double variable by the iteration of the outer for loop times 2 + 1
update the temp variable to 1 / temp
set a if statement to check if values are smaller than epsilon
if true, break
end of if statement
update the total to itself plus temp variable
update counter, start and hold variables
end of outer for loop
update total to itself times sqrt(12)
return total
```

For the pi_madhave_terms, return static variable

2.3 euler.c

```
import all files
set a static counter variable
set a total,start, and hold variables to 0
use a while loop to run until it hits smaller than epsilon
set a temp variable to 1 divided by the start times start
set a if statement to check if values are smaller than epsilon
if true, break
end of if statement
update the total to itself plus temp variable
update counter, start and hold variables
end of for loop
update the total amount to itself times 6
update the total amount to the newton(itself)
update counter variable to itself plus one
return total
```

For the pi_euler_terms, return the static variable.

2.4 viete.c

```
import all files
set a static counter variable
set a total amount to newton(2)/2
set a ”old” double variable to newton(2)
use a while loop to run until it hits smaller than epsilon
set a ”new” double variable to newton(old + 2)
set a if statement to check if values are smaller than epsilon
if so set watch to 1
end of if statment
set new to itself divided by two
update the total amount to itself times new
update counter variable to itself plus one
end of while statement
return total
```

For the pi_viete_factors, return the static variable.

2.5 newton.c

pseudocode provided on Assignment document

For the sqrt_newton_iters, I will just have a counter variable like I did in my other functions and update that by one as the computation runs.

2.6 bbp.c

```
import all files
set a static counter variable
set a total,start, and hold variables to 0
use a while loop to run until it hits smaller than epsilon
set a ”left” double variable to 16
use a for loop to loop from 0 to the iteration of start
update left variable to itself times 16
end of inner for loop
update left to 1 divided by itself
set ”top” double variable to the iteration of the outer for loop times (120 times iteration of the outer loop plus 151)
update top variable to itself plus 47
set ”bottom” double variable to the iteration of the outer for loop times 512 plus 1024
update bottom to itself times the iteration of the outer for loop and plus 712
update bottom to itself times the iteration of the outer for loop and plus 194
update bottom to itself times the iteration of the outer for loop and plus 15
set temp variable to top divided by bottom
set temp variable to itself times left
set a if statement to check if values are smaller than epsilon
if so set watch to 1
end of if statment
update the total to itself plus temp variable
update counter, start and hold variables
end of outer for loop
return total
```

For the pi_bbp_terms, return the static variable.

2.7 mathlib-test.c

```
include all header files
create main function and parse through arguments
for each case set a Boolean to True
use if statement run through test functions and update Boolean variable as such depending on outcome
```

3 Files

bbp.c- A header file containing two functions of BBP formula to approximate pi and amount of computed terms.
e.c- A header file containing two functions of Taylor series to approximate e and amount of computed terms.
euler.c- A header file containing two functions of Euler's solution to approximate pi and amount of computed terms.
madhava.c- A header file containing two function of Madhava series to approximate pi and amount of computed terms.
viete.c- A header file containing two functions of Viete's formula to approximate pi and amount of computed factors.
newton.c- A headerfile containing two functions of Newton's method to approximate a square root and amount of computed factos.
mathlib.h- The interface of my math library.
mathlib-test.c- The main file(function) that tests my math library functions.
Makefile- This allows us to use clang and compile our program.
README.md- In markdown format, it tells us how to run the program and how the program was made.
DESIGN.pdf- This is how I started thinking about how to code the program.
WRITEUP.pdf- A PDF considering graphs made by UNIX tool displaying the difference between the values reported by my function and the math's library.

4 Credit

1. I was given explanations and formulas for the functions via the Assignment 2 document.
2. Professor Long gave us the pseudocode for the Newton-Raphson method in python format that I used and turned into C.
3. I used some of the code that TA Sloan gave us during section on 10/06 for the Makefile.
4. I used the resources provided in the resources folder in the ASGN2 file.

5 Error Handling

1. I had realized that my definition/usage of epsilon was wrong so I had to entirely change that part of logic in my code in essentially header file. I ended up usually having a hold/old variable so I could do new value minus the hold variable and see if that is smaller than epsilon.
2. I had previously thought that we were passing in arguments into our functions but that is not the case.
3. For a lot of my functions where the summation starts at 0, I realize I needed to add a test case for most of them because of the oddity that 0 provides into a mathematical equation.
4. In my pseudocode that I had written before actually coding, I had thought I would use for loops. After actually coding, I realized that things would be easier if I use a while loop that keeps running until I hit the epsilon mark.
5. For my e.c file, I realized that for my if statement of checking when to stop running the function, I need to add another conditional statement as it bugs out a bit at 0.
6. In terms of doing exponentials, I had thought that I would need to use a for loop and each time the for loop ran I would have to multiple the number by itself. This was an error in logic because I needed to multiply the number by the original number itself. To exemplify, what I did was 3 x 3 = 9 would then go to 9 x 9 which is clearly incorrect.
7. Honestly, for the euler pseudocode that I had provided, I don't know exactly what I was doing. I read back on it and completely changed it after thinking about it more indepthly.
8. My viete function worked perfectly, but I had one more computation than the example which obviously makes my value more accurate.
9. I realized that I had to be returning totals in my calculation functions.
10. I realized that the functions that returns the amount of computations uses int type variables.