Assignment Two Design.pdf

By James Yim

Description of Program:

This program will have several different aspects to its purpose. The first part of the program is that we will create a library of certain mathematical functions. The functions that we will be adding are: Exp(), Sin(), Cos(), Sqrt(), Log(). The second part of the program will integrate specific functions using Simpson's rule.

Files to be included in asgn2

- functions.c
  - This file contains the code which will be the functions that we will integrate in integrate.c
- functions.h
  - This file is a header file for the functions that we will integrate
- integrate.c
  - performs integrations based on user input on a set amount of functions
- mathlib.c
  - This file contains the code of the math library that we created.
- mathlib.h
  - header files for the math library functions of mathlib.c
- Makefile
  - This file directs the compilation process of the program
- README.md
  - This file will be in Markdown syntax and describes how the program works. Also will describe the command line options.
- DESIGN.pdf
  - This file will describe the design of the program and include pseudocode
- WRITEUP.pdf

- This will include graphs showing the integrated values with the various amount of partitions using GNUPLOT. It will also have an analysis of the produced graphs.

Design & Pseudocode

Mathlib.c

Exp()

```
exp(x)
{
        termSum = 1
        seriesSum = termSum
        k = 1
        while (termSum > epsilon) //where epsilon is a small enough number
        {
                termSum *= |x| / k
                k += 1
        }
        if x > 0 return termSum
        else x < 0 return 1 / termSum
}
```

Sin()

```
sin(x)
{
seriesValue = x
currentSeriesValue = 1
termValue = x
K = 3
        while |termValue| > epsilon
        {
                termValue = termValue * (x * x) / k * (k -1)  This gives us a factorial
                currentSeriesValue = -currentSeriesValue //this switches the sign
                seriesValue += currentSeriesValue * termValue
                k += 2
        }
        return seriesValue
}
```

Cos()

```
cos(x)
{
currentSeriesValue, seriesValue = 1
termValue = x
K = 2
        while |termValue| > epsilon
        {
                termValue = termValue * (x * x) / k * (k -1)  This gives us a factorial
                currentSeriesValue = -currentSeriesValue //this switches the sign
                seriesValue += currentSeriesValue * termValue
                k += 2
        }
        return seriesValue
}
```

Sqrt()

```
sqrt(x)
{
        currentTerm = 0
        nextTerm = 1
        while |nextTerm - currentTerm| > epsilon
        {
                currentTerm = nextTerm
                nextTerm = .5 * (currentTerm + x / nextTerm)
        }
        return nextTerm
}
```

Log()

```
log(x)
{
        seriesSum = 1
        p = exp(seriesSum)
        while | p - seriesSum | > epsilon
        {
                seriesSum = seriesSum + x / p - 1
                p = exp(y)
        }
return seriesSum
}
```

integrate()

```
integrate(f(x), upperBound, lowerBound, numberOfPartitions)
{
	h = (upperBound - lowerBound) / numberOfPartitions)
	seriesSum = f(lowerBound) + f(upperBound)
	for (i = 2; i < numberOfPartitions; i += 2)
	{
		seriesSum += 2 * f(lowerBound + i * h)
	}
	for (i = 1; i < numberOfPartitions; i += 2)
	{
		seriesSum += 4 * f(lowerBound + i * h)
	}
	Return (seriesSum *= h / 3)
}
```

Integrate.c

Choosing which function to integrate:

```
Receive options from user using getopt()
{
	Receive either a, b, c, d, e, f, g, h, i, or j to decide which function
	Receive an optional n which represents the amount of partitions to integrate.
	Receive both p and q which represents low and high ends of the integral
	Receive H which shows how to use integrate.c
	{
		If (option is a function set functionCase to true)
		If (option is n, set number of partitions to number that was inputted)
		If (option is p, set lowerBound to number that was inputted)
		If (option is q, set upperBound to number that was inputted)
		If (option is H display how to use integrate.c)
		If (no option is inputted give error)
	}
}
```

```
Displaying integrals
{
        Check which options a though i are true
        If (true)
        {
                print(f(x), lowerBound, upperBound, numberOfPartitions)
                Loop through the amount of partitions printing out the integrals of each partition.
        }
}
```

Notes about code

EPSILON

Epsilon is used as a way of noting very small changes to a value. We set EPSILON to 10^-14. We stop iterating if a value gets smaller than this number as adding it will not make much of a change

getopt()

getopt() allows us to get user input from a set of inputs we set. In our code our set of inputs range from *a* to *i* for different functions and *n, p, q* for number of partitions, lower bounds and upper bounds respectively. Lastly we have *H* which provides a help guide for the options available.

switch()

switch() allows us to pick which cases we want to have. We know which cases we want based on input from getopt()

Error Handling

What happens if a user input an illegal option?

The way that I handled this possibility is by using *default:* when it comes to cases. I print out an error message saying that an illegal input was given and I just exit the program.

What happens if a user does not input any option?

When this happens the program just quits.

Credit

- I attended Eugene's section on 1/14/22. He helped with Makefile, and getopt(). Also gave general advice on how to go about the assignment.
- I attended Audrey's section on 1/19/22. They clarified how precise our final integral answers have to be.
- The class Discord helped with general questions I had