Despina Patronas

Asgn2 Design Document

Pre-lab Part 1:

1) **<u>Psuedo-code to approximate e^x:</u>** //will follow the Maclauren series formula

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

```
//To do so, will use helper functions for numerator (raise base b to power p) and denominator (factorial)

//Function for calculating a base b input to an input power p

real power ( real b, int p)

        Define res set to 1

        Iterate through (0)  to (p -1)

                Total = total * b

        Return total

End power


//Function to calculate the factorial of input x

real factorial (real n)

        If n == 0 or n == 1

                Return 1            //both 0! = 1 and 1! = 1

        End if

        Else

                Define real total set to 1

                While n > 1

                        total = total * n

                        Decrement n

                End while

                Return total

        End else

End factorial
```

```
//function to approximate exponential number

real Exponential(real x)

        If x == 0                   //e^0 = 1

                Return 1

        End if

        Else

                Define real threshold = 10^-9     // constant threshold value

                Define int order = 2

                Define real result = 1 + x          //this takes care of the 0st and 1st terms

                Define real difference = 1          //this tells us when to stop approximating terms set to 1

                                                    //because we will be comparing it to threshold

                While difference > threshold

                        Define real prev_Term set to result                         //store our (current  order-1) term

                        Result = result + ( power(x, order) / factorial (order) )   // add the current term to last term

                        Define real next_Term set to result                         //store our current order term

                        Difference = | prev_term – next_Term |                      //absolute value of the two order terms

                        Increment order

                End while

                Return result

        End else

End Exponential
```

2) <u>Pseudo-Code for Printing the output of e^x</u>

```
//we will be printing e^x from [0,9] in steps of 0.1

//include the step, the exponential function, the math library exponential function, and the difference

Void print_Results_Exp ()

    Iterate through 0 to 8 in steps of 0.1

    Define and set real difference = | call exponential(iteration) - call math_lib_exp(iteration |
```

Display banner message for the 4 columns listed above

Display the iteration number, call exponential(iteration), call math_lib_exp(iteration), difference

End print_Results_Exp

Pre-Lab Part 2:

1) **<u>What does getopt() return</u>**

   A: If an option character passed through the main function that match a character in optstring will be caught and returned by getopt() function. (this results in a pointer to external variable optarg)

   -1 If no options to process

   ? if unrecognized option and stores into optopt

   : if an option requires a value and none is given (when colon is placed as first char of options string)

2) **<u>Is a `bool` or an `enum` the best choice? Explain why.</u>**

   A. Depends on what needs to be accomplished. Bool is simple and straightforward however.

3) **<u>Provide pseudocode for your main function (assume helper functions are available to you)</u>**

```
Int main ( int argc, char **argv)

    Define OPTIONS "sctea "
    Declare and initialize choice = 0

    While (choice = getopt (argc, argv, OPTIONS) != -1
        Case based on choice
            //test sine helper function
            Case 's'
                Test_sin()
                break

            //test cosine helper function
            Case 'c'
                Test_cos()
                break
            //test tangent helper function
```

```
                Case 't'
                        Test_tan()
                        Break

                //test exponential helper function
                Case 'e'
                        Test_exp()
                        Break

                //test all helper functions
                Case 'a'
                        Test_sin()
                        Test_cos()
                        Test_tan()
                        Test_exp()
                        Break

        End Case

    EndWhile

    If (argc == 1 )

        Display error message

        Return -1

    End if

    Return 0

End Main
```

<u>//Rest of Design Implementation for Assignment 2</u>

<u>//Calculations</u>

//calculations follow the asgn2 lab using Horner Normal Form centered around 0
//14th term series

It is a lot easier to square a number than to raise it to a power, so we can simplify it by putting the formula into *Horner normal form*, by factoring out x as much as possible:

$$\sin(x) \approx \frac{x((x^2(52785432 - 479249x^2) - 1640635920)x^2 + 11511339840)}{((18361x^2 + 3177720)x^2 + 277920720)x^2 + 11511339840}.$$

Why Horner normal form? It has the fewest multiplications.

```
Real _Sin( real x)

    Declare real numerator = 0

    Declare real denominator = 0
```

Declare real x2 = power(x,2)

Numerator = $x((x^2(52785432-479249x^2)-1640635920)x^2+11511339840)$

Denominator = $((18361x^2+3177720)x^2+277920720)x^2+11511339840$

Return numerator / denominator

End _sin

Consider the corresponding approximant for $\cos(x)$ centered around 0 written in Horner normal form:

$$\cos(x) \approx \frac{(x^2(1075032-14615x^2)-18471600)x^2+39251520}{((127x^2+16632)x^2+1154160)x^2+39251520}.$$

//calculations follow the asgn2 lab using Horner Normal Form centered around 0
//14th term series

Consider the corresponding approximant for $\cos(x)$ centered around 0 written in Horner normal form:

$$\cos(x) \approx \frac{(x^2(1075032-14615x^2)-18471600)x^2+39251520}{((127x^2+16632)x^2+1154160)x^2+39251520}.$$

Real _Cos (real x)

Declare real numerator = 0

Declare real denominator = 0

Declare real x2 = power(x,2)

Numerator = $(x^2(1075032-14615x^2)-18471600)x^2+39251520$

Denominator = $((127x^2+16632)x^2+1154160)x^2+39251520$

Return numerator / denominator

End _Cos

//since our domain is restricted from [-pi/3, pi/3] incremented by pi/16 steps

//tan = sin/cos is no longer at risk of becoming undefined at pi/2

Real _Tan (real x)

Return Test_Sin(x) / Tes_Cos(x)

End _Tan

//Testing / Output

//It is a good idea to define pi for calculations on the following

//Printing results functions are created similarly but with their respective trig functions

//note Sin and Cos testing range:[-2pi, 2pi] while tan range: [-pi/3, pi/3] and all with pi/16 steps

//sin

Void print_results_sin ()

Iterate through –[2pi, 2pi] in steps of pi/16

Define and set real difference = | call _sin(iteration) - call math_lib_sin(iteration |

Display banner message for the 4 columns listed above

Display the iteration number, call sin(iteration), call math_lib_sin(iteration), difference

End print_results_sin

//cos

Void print_results_cos ()

Iterate through –[2pi, 2pi] in steps of pi/16

Define and set real difference = | call _cos(iteration) - call math_lib_cos(iteration |

Display banner message for the 4 columns listed above

Display the iteration number, call cos(iteration), call math_lib_cos(iteration), difference

End print_results_cos

//tan

Void print_results_tan ()

Iterate through –[pi/3, pi/3] in steps of pi/16

Define and set real difference = | call _tan(iteration) - call math_lib_tan(iteration |

Display banner message for the 4 columns listed above

Display the iteration number, call cos(iteration), call math_lib_tan(iteration), difference

End print_results_tan