Assignment 2 Design – Jack Vento (1816097)

Prelab Part 1

1. Write pseudo-code for approximating e^x with either a for or while loop.

As the lab manual describes, this function approximates the Taylor series by recursively adding x/n in a while loop until we reach the desired error level (Epsilon).

```
Begin
ans ← term  ← 1.0
n ← 1
while term > EPSILON (10^-9)
    term ← term * x / n
    ans ← ans + term
    n ← n +1
return ans
End
```

2. Write pseudo-code for printing the output for e^x.

```
Begin
for i <= 9; i ← i + 0.1
    x ← Exp(i)  // Our version
    y ← exp(i)  // Standard C version
    diff ← x - y
    Print i, x, y, and diff using the formatted printf in the lab manual
End
```

Prelab Part 2

1. What does getopt() return? Hint: check the man page.

Getopt() returns -1 once it is out of options to parse and an error code 0-4 otherwise. It returns 0 when it successfully parses the command-line options, 1 if there are errors in its execution, 2 if it doesn't understand the passed parameters, 3 if an internal error (i.e. out-of-memory) occurs, and 4 when it is called with -T (helper parameter to test if getopt(1) is an enhanced or an old version).

2. Is a bool or an enum the best choice? Explain why.

An enum is the best choice since the command-line options are mutually exclusive and can only be chosen one at a time. Since this is the case, an enum allows us to have the least amount of variables (1 enum vs 5 bools), smallest memory footprint (1 enum = 4 bytes vs 5 bools =5 bytes) and not lose any information. Were the program to allow multiple inputs, a bool per option would be the best choice since an enum would not allow us to track each individual option. However, this is not the case, so an enum is the most efficient choice.

3. Provide pseudo-code for your main function. Assume you have helper functions available to you.

Start
Declare input_arg enum to be set in the while loop
Check if there are more or less than 1 command-line arguments supplied, print an error and return if so.
while there are options to parse (getopt != -1)
    switch (getopt)
        input_arg ← enum corresponding to the supplied command-line option (ex: input_arg = sin)
switch input_arg
    call sin(), exp(), tan(), exp(), or all of them depending on the supplied enum
End

Library Functions Design

Sin/Cos/Tan

These functions will give a rudimentary approximation of the trig functions using n=14 terms. If they end up having somewhat unsatisfactory accuracy, I'll instead use a Maclaurin series with a strict epsilon, raising the amount of terms involved.

Begin
if input is not between [-2π, 2π]
    Convert input to be between [-2π, 2π] using unit circle
squared ← x^2
return Pade approximant (see lab manual for Horner normal equation)
End

Sin/Cos Tests

See Prelab Part 1.2. Functions will be the exact same with exp/Exp swapped for the appropriate trig functions and the for loop ranging from [-2π, 2π] with steps of π/16.

Tan Test

See Prelab Part 1.2. Function will be the exact same with exp/Exp swapped for tan/Tan and the for loop ranging from [-π/3, π/3] with steps of π/16.