PRELAB PART 1

Pseudocode for e^x

- while(answer>0.00000001….) {

  answer=x^n/factorial(n)
  n=term;x=value
  sum+=answer

Pseudocode for printing e^x

- for (i=0;i<=10;i++) {
  exp(i)
  print(i)

PRELAB PART 2

1.getopt() returns the command argument that you time in terminal after you run .c

2.Enum would be the best choice because you can label all your inputs and determine where to go when user inputs.

3.

Main(TAKE IN ARGUMENTS FROM COMMAND LINE) sctea only

```
while(opt=getopt(int argc, char argv)!=-1) {
        Switch(opt)

                Case s:
                        approx=Sin(x) -created function
                        Answer =sin(x) -library answer
                        Difference =approx-answer
                        print (Label and value)
                        Break;
                Case c:
                        approx=Cos(x) -created function
                        Answer =cos(x) -library answer
                        Difference =approx-answer
                        print (Label and value)
                        break;
                Case t:
                        Call sin(x) and cos(x)
                        approx=sin(x)/cos(x)
                        Diff
                        print(label and values)
                        Break;
                Case e:
                        e(x)
```

```
        for(i=0;i<10;i++)
        print(label, answer)
    Case a:
        Print exactly from case s,c,t,e
    Case:?
        break;
```

PSEUDOCODE

1.Define PI value;
2.Create Sin(x);Cos(x);Tan(x);Exp(x); function

(Taylor series that uses odd terms)
For case s:
Sin(x) {
If x is greater than PI, then we minus 2*PI, so more accurate
If x is less than -PI, then we add 2*PI

for (i=0;i<10;i++) {
Sign=Power(-1,i)
Power(x,2*i+1)
f=Factorial(2*i+1)
}

Return sign*power/Factorial

(Taylor series that uses even terms)
For case c:
Cos(x) {
If x is greater than PI, then we minus 2*PI, so more accurate
If x is less than -PI, then we add 2*PI

for (i=0;i<10;i++) {
Sign=Power(-1,i)
Power(x,2*i)
f=Factorial(2*i)

Return sign*power/Factorial

Case t:
Tan(x) {
Go into Sin(x) and Cos(x)
Return Sin(x)/Cos(x)

Case e:
Exp(x) {
Define a term that equal 0.000001 (really low number)
i=1
e^x=Power(x,i)/Factorial(i);
i+=1
sum+=e^x
Return sum

I created a function to solve factorials
Factorial(x) {
If x is 0 or 1 then return 1
Else:
answer=x*Factorial(x-1) RECURSION

for powers
Power(x) {
If x=0 return 1
Else:
for(i=1;i<x;i++){
answer*=x

Then go back to main function and make print statements to print

Taylor Series works by adding the next terms.