Nicolas Ayala
nmayala@ucsc.edu
Design for assignment 2: Math library

Pre-Lab 1
1)

```
double Exp(double x){
        double answer = 1;                  //Begin with the 0th term
        double current_term = 1;       //0th term is 1
        //i is the current term number
        for(int i = 1; current_term > EPSILON; i++){         //Until the terms get below
                                                              //Epsilon, keep adding
                                            //terms
                current_term*=x;        //The next term is calculated by taking the
                current_term/=i;        //old term and multiplying it by x/i
                answer+=current_term;//Add this term to the sum
        }
        return answer;
}
```

2)

```
double PrintExp(){
        print("x           Exp             Library    Difference\n");
        print("-           ---             --------     ------------\n");
        for(double x  =  0; x <= 10 ; x += 0.1){
                print(truncate_to_five_digits(x)+"   "
                        +truncate_to_five_digits(Exp(x))+"      "+
                        truncate_to_five_digits(exp(x))+"  "+
                        truncate_to_eight_digits(Exp(x)-exp(x)) );
        }
}
```

Pre Lab part 2
1) getopt(argc, argv, "optns") returns another option found in the string "optns" as a character, unless there is no more options, then it returns -1.

For example: when a C executable is called and followed by options such as -c -d -f, the first call of get opt will return 'c', the next call will return 'd', the next one will return 'f', the next one will return -1.
2)
For such a simple program with only 5 options and a simple relation between the five, namely that only the first is to be considered, then using bools is ok. The upsides of using enums rather than bools is readability for programmers, but has little effect on the output and performance of

the code. The downside of using enum is that I can't directly use logical connectives, and so to simulate the logical relations of these options would require converting to booleans anyway. In the case of this program, using enums may actually decrease readability, as the relations between these options will be ever so slightly more convoluted.

3)

```
main(argc, **argv){
	bool test_sin_flag = false;
	bool test_cos_flag = false;
	bool test_tan_flag = false;
	bool test_exp_flag = false;
	bool test_all_flag = false;

	while(Theres_more_options()){
		c = next_option();
		switch(c){
		case 's':
			test_sin_flag = true;
		break;
		case 'c':
			test_cos_flag = true;
		break;
		case 't':
			test_tan_flag = true;
		break;
		case 'e':
			test_exp_flag = true;
		break;
		case 'a':
			test_all_flag = true;
		break;

		}
	}
	if(test_sin_flag){
		test_sin();
		return 0;
	}
	if(test_cos_flag){
		test_cos();
		return 0;
	}

	if(test_tan_flag){
```

```
            test_tan();
            return 0;
        }

        if(test_exp_flag){
            test_exp();
            return 0;
        }

        if(test_all_flag){
            test_sin();
            test_cos();
            test_tan();
            test_exp();
            return 0;
        }
        return 0;
}
```

Design of Program

There will be a function called restrict_to_circle, that maps a double into the range (-pi, pi) such that it represents the same radian

```
        double restrict_to_circle(double x){
            x -= PI;
            x = double_modulus(x, 2PI);
            x+=PI
            return x;
        }
```

There will be a function called pade_of_sin, which takes as input a double x

x is first restricted to the circle with restrict_to_circle.

Then the following Padé approximation of sin is applied to x:

$$\sin(x) \approx x\frac{(x^2\,(52785432 - 479249x^2) - 1640635920)x^2 + 11511339840}{((18361x^2 + 3177720)x2 + 277920720)x^2 + 11511339840}$$

There will be a function called pade_of_cos, which takes as input a double x

x is first restricted to the circle with restrict_to_circle.

Then the following Padé approximation of cos is applied to x:

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

There will be a function called Sin, which takes as input a double x

The following is set theory terminology, not code terminology
The following four regions are a partition of the Reals (they are pairwise disjoint and their union is the Reals)
Region 0 = union of {[-pi/4,pi/4) + k*2pi, where k is an integer}
Region 1 = union of {[-pi/4,pi/4) + pi/2 + k*2pi, where k is an integer}
Region 2 = union of {[-pi/4,pi/4) + pi + k*2pi, where k is an integer}
Region 3 = union of {[-pi/4,pi/4) + 3pi/2+ k*2pi, where k is an integer}

```
if(x is in Region 0)
    return pade_of_sin(x)
if(x is in Region 1)
    return pade_of_cos(x - pi/2)
if(x is in Region 2)
    return -pade_of_sin(x - pi)
if(x is in Region 3)
    return -pade_of_cos(x - 3pi/2)
```

There will be a function called Cos, which takes as input a double x
Cos simply returns Sin(pi/2 - x)

There will be a function called Tan, which takes as input a double x
x is first restricted to HALF of the circle, i.e. the range (-pi/2, pi/2)
Then the following Padé approximation of tan is applied to x:

```
tan(x)≈ ((((x^2 - 990)*x^2 + 135135)*x^2 - 4729725)*x^2 + 34459425)*x
    /((((((x^2 - 308)*x^2 + 21021)*x^2 - 360360)*x^2 + 765765)*45)
```

There will be a function called Exp, which takes as input a double x
Implementation of Exp can be seen in Pre-Lab 1, 1

There will be functions called test_sin, test_cos, test_tan, test_exp
The function test_fnc will compare the implementation of fnc in this program with the implementation of fnc found in math.h
It will iterate over a set of values, and for each value x it will output:

```
x
Fnc(x)          //the implementation in this program
fnc(x)          //the implementation in math.h
Fnc(x)-fnc(x)   //the difference between the two
```

There will be a main function
It looks for the options -s, -c, -t, -e, and -a, that are passed along side it as the program is called. The -s option runs test_sin, the -c option runs test_cos, the -t option runs test_tan, the -e option runs test_exp, the -a option runs all of the tests. In the case of multiple flags, the first test is run and then the program terminates. In the case of no flags, the program terminates with no input. Instead of using booleans, I simply allowed the program to run the first test it sees and then terminate.