Math Functions Without <cmath>

Recall the Babylonian Algorithm from chapter 2:

Algorithm 1.1 The Babylonian Algorithm for Computing the Square Root of 2

1. Input A
2. Set x = A/2.
3. Replace x with the average of x and A/x.
4. Repeat Step 3 until its effect upon x is insignificant.
5. Return x.

This is easily adapted to compute the square root of any real number, like so:

```
//G. Hagopian
//Implement the babylonian algorithm

#include <iostream>
using namespace std;

// prototype for the root function
double root(double);

int main()
{
	double A; // The number whose root we seek
	cout << "\nEnter a number to find its square root: ";
	while(cin >> A)  // Use the EOF character ctrl+D to quit
	{
		(A>=0) ? cout << root(A) << endl
		       : cout << "i*" << root(-A) << endl;
	}
}

double root(double A)
{	// Use the babylonian algorithm to compute the square root
	double x = A, temp = x/2;
	const double tolerance = 1e-14;
	while(temp - x > tolerance || x - temp > tolerance)
	{
		x = temp;
		temp = (x + A/x)/2;
	}
	return temp;
}
```

Here is output from a typical run of this program:

```
Enter a number to find its square root: 5
2.23607
6
2.44949
7
```

```
2.64575
8
2.82843
-9
i*3
^D
Press any key to continue . . .
```

Algorithms for computing other math functions often implement various approximating polynomials such as

$$\sin(x) \approx \sum_{n=0}^{N} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{6} + \frac{x^5}{120} - \cdots + \frac{(-1)^N x^{2N+1}}{(2N+1)!}$$

$$\cos(x) \approx \sum_{n=0}^{N} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2} + \frac{x^4}{24} - \cdots + \frac{(-1)^N x^{2N}}{(2N)!}$$

$$\exp(x) \approx \sum_{n=0}^{N} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots + \frac{x^N}{N!}$$

$$\ln(x+1) \approx \sum_{n=1}^{N} \frac{(-1)^{n+1} x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + \frac{(-1)^{N+1} x^N}{N}$$

Your task in this lab is to implement a function for computing

1. $\exp(x)$ (Note: In theory, the sum converges for all *x*. In practice, there are limits…)
2. ln(*x*) (Note: In theory, the sum converges only for $0 < x \leq 2$.)

Use a driver function to test that exp(*x*)exp(*y*) = exp(*x*+*y*) and ln(*x*)+ln(*y*) = ln(*xy*)