Practice Midterm

Background:

The C++ function `exp(x)` can be approximated by the Maclaurin Series:

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

This series will always converge, but not very quickly.

The above infinite series can be re-expressed so that it is easier to program as follows:

1. $t_0=e_0=1$
2. $t_{n+1}=t_n\frac{x}{n}$,
   $e_{n+1}=e_n+t_{n+1}$

Assignment:

Write a template function named MacExp(T x, T Tol) :

1. T is the template type.
2. x is the exponent.
3. Tol is the error tolerance. It should default to 10^{-6} . When two successive e_n differ by less than Tol, iterations stop and the function returns the last e_n

Write a driver main functions that computes e^0 using the default tolerance and `double` arithmetic. The main program should also compute e^1 using a tolerance of 10^{-12} and `long double` arithmetic.

Check your answer using the C++ builtin function `exp(x).`

Console Log:

```
MacExp<>(0.0): 1, std::exp(0.0): 1
MacExp<>(1.0L): 2.718281828459045, std::exp(1.0L): 2.718281828459045
e ~= 2.71828182845904523536028747135 27 ...
```

Solution:

```
// Maclaurin.h
#include <cmath>
using std::exp;

template<typename T>
T MacExp(T x, T Tol=T(0.0000001))
{
  T tn{T(1.0)};
  T en1{tn};
  T n{T(0.0)};
  T en;
  T Diff;
  do
  {
    // Last en1 becomes the new en for this iteration.
    en = en1;

    // Compute next term.
    tn *= x / ++n;

    // Add term to sum.
    en1 = en + tn;

    // Compute absolute value of difference between last two terms
    Diff = en1 - en;

    // In the midterm you must use the correct C++ absolute value function
    // from the library.  I do absolute value this way to avoid using any function.
    Diff = Diff < T(0.0) ? -Diff : Diff;
  } while(Diff > Tol);

  // Return last en1, which is best approx of exp(x).
  return en1;
}

// main.cpp
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setprecision;

#include "Maclaurin.h"

int main(void)
{
  // First compute exp(0.0) using double arithmetic and default tolerance.
  // We should expect exp(0.0) = 1.0.
  {
    auto macexpx = MacExp<>(0.0);
    cout << setprecision(16) <<
      "MacExp<>(0.0): " << macexpx <<
      ", std::exp(0.0): " << exp(0.0) <<
      endl;
  }

  // Second compute exp(1.0L) using long double arithmetic and tolerance 10^-16.
  // We should expect exp(1.0) to be close to e ~= 2.7182818284590452353602874713527 ...
  {
    auto macexpx = MacExp<>(1.0L, 1.0e-16L);

    cout << setprecision(16) <<
      "MacExp<>(1.0L): " << macexpx <<
      ", std::exp(1.0L): " << exp(1.0L) <<
      endl;

    cout << "e ~= 2.7182818284590452353602874713527 ..." << endl;
  }

  return 0;
}
```