CS 507 Computing Foundations
Homework #1

Tyler Phillips

October 5, 2017

Problem #1 (10pts)

Write programs to compute number e accurate to 8 decimal places, using standard C++ basic concepts of function, recursion, types, control flow and so on. (This is generally the style you'll be asked during your online test, phone/on-site interview and so on in future). NOTE: This is the background and ideas we can use to break down the problem into applicable steps. By definition, the exponential function is

$$e^x = \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n$$

$$e^x = \sum_{n=0}^{\infty}\frac{x^n}{n!}$$

1.1: (2pts)

What is the output of the above program? Is it correct? Explain why with detailed analysis. (Hint, review roundoff error you should have learned from "programming" course or introduction to computation course youve taken).

1.2: (8pts)

Implement the 2nd method listed above to compute number e accurate to 8 decimal places (7 pts). It would be nicer if you can write a subroutine (or more if you want. We wont limit your creativities) called by the main function to compute each term in the series, i.e., $a_n = \frac{x^n}{n!} = \frac{x}{n}a_{n1}$(1pt). (Hint: check the reference book or others for the concept of recursion involving a function calling itself. An example code on computing n! can be found by googling recursion factorial c++).

Problem 1 File tree:

```
Phillips_Tyler_HW_01
└ problem_1
   ├─ makefile
   ├ src
   │  ├─ main.cpp
   │  └─ exponential.cpp
   └ include
      └─ exponential.h
```

Compile and run:

From problem_1 directory:
$ make – will compile and give executable
$ make run – will compile, run, and clean

1.1)

Output:

```
****************************
Problem 1.1
****************************
d = 1e-308
exp(1) = 1
```

This output is not correct this is because there is roundoff errors in the computations. Roundoff errors are the difference in the actual number and the computed number due to using a computer with a finite number of values. The issue for our problem is that when we are trying to add (1 + 1e-308) we are getting a value of 1. Therefore, if we have 1 to any power no matter how large we will always end up with 1.

1.2)

Output:

```
****************************
Problem 1.2
****************************
i:     e(1):
1       2.000000000
2       2.500000000
```

```
3        2.666666667
4        2.708333333
5        2.716666667
6        2.718055556
7        2.718253968
8        2.718278770
9        2.718281526
10       2.718281801
11       2.718281826
12       2.718281828
exp(1)  2.718281828 using cmath library
```

For this problem I have created functions for the numerator (power) and denominator (factorial). However, it should be noted that this is not the most efficient way to calculate this problem. We would want to keep the current factorial and power value and only do a single multiplication each step by the new n value.

Problem #2 (10pts)

Write programs to define your own Complex class for calculations of complex numbers including +, -, *, = and modulus $|x|$ operations. Make sure the calculations are compatible to real numbers. Enclose your Complex class definition in a complex.h header file. Suppose we have 2 complex numbers: a = (2.0,2.0i),b = 5.0, in the main program, output the following calculation

$$c = a + b, \quad d = a - b, \quad |c|$$

Helpful information. Take a look at example codes in Lec04, page 15, Lec05, page 13. First, use class keyword to define a Complex class, which should include 2 variables: re and im representing real and imaginary parts. Second, make constructors and destructor based on the above 2 variables. The constructor should include at least 1 argument. Third, define several member functions that can return real part, imaginary part, modulus of a Complex object. Fourth and most importantly, overload +, -, *, = operators. In our example code in Lec04, we used the following operator overloading:

void operator ++ () ++count;

where void is the return type, operator is the keyword, ++ is the increment operator to be overloaded, () includes empty argument. The function body has no return value (thats why a void was used) but the internal variable count would be increased by 1. Similarly, here we would define another return type and return both real and imaginary part of a Complex object. The argument in () is not empty any more but should be another Complex class to be added, subtracted, multiplied, equaled. Last, use class template so that the Complex class can be used on any data type.

Problem 2 File tree:

```
Phillips_Tyler_HW_01
└problem_02
  ├─makefile
  ├src
  │ └─main.cpp
  └include
    └─complexNum.h
```

Compile and run:

From problem_2 directory:
$ make – will compile and give executable
$ make run – will compile, run, and clean
Output:

```
********************************
Problem 2
********************************
(2,4i) + (5,0i) = (7,4i)
(2,4i) − (5,0i) = (−3,4i)
(2,4i) ∗ (5,0i) = (10,0i)
(2,4i) / (5,0i) = (0.4,infi)
|(7,4i)| = 8.06226
```