HW3
Out: October 3, Tuesday -- DUE: October 11, Wednesday, 12:00pm
EC327 Introduction to Software Engineering – Fall 2023

Total: 100 points

- *Comment your code and use a clean, easily understandable coding convention. A few coding style guides for your reference:*

http://geosoft.no/development/cppstyle.html
https://google.github.io/styleguide/cppguide.html

Submission: ***Failure to follow the guidelines WILL result in loss of points on the assignment.***

1. Completed assignments **must** be submitted on GradeScope.
2. Write your answers clearly in your favorite text editor, and submit a .**cpp** file for each question (Q1.cpp, Q2.cpp, Q3.cpp) ZIPPED into a file named: **BUusername_HW3.zip**
   For example: **acoskun_HW3.zip** -> Has Q1.cpp Q2.cpp Q3.cpp
3. Please make sure your code compiles and runs as intended on the **engineering grid**. Code that does not compile on the eng-grid will NOT be graded and will receive a 0.

Late Homework Submissions:
Recall the late submission policy and the fixed penalty of 20% for submitting up to two days after the deadline. Also recall that we only grade your latest submission (e.g., if you submit both on time and after the deadline, only your late submission is graded).

Please write C++ code for solving the following problems.

Q1. *Functions and basic recursion.* [30 points]
The exponential function e^x can be approximated using the following (infinite) Taylor series expansion:

$$1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\ldots$$

a. [15 points] Write a recursive function with the following prototype to compute the sum of the Taylor series above. The function takes an input parameter n and computes the expansion as a sum of the first n terms only.

```
double ExpTaylor(int x, int n);
//computes the sum of the Taylor series with n terms
/* e.g., for n=3, the function should compute 1+ (x / 1!)+(x^2 / 2!) for the given x */
```

b. [10 points] Write a test program that asks the user to enter integers x and n, and computes e^x using the `ExpTaylor` function. Assume the user will always enter integers x and n, where x>=0 and n>0. The program prints the result of the function on the console.

c. [5 points] In your program, compute e^x using the cmath exp function (e.g., exp(5) is e^5) for a given x, and then write code that finds the minimum n for the ExpTaylor function such that the exp function result and ExpTaylor result differs by at most 0.01. The program prints n, exp result, and ExpTaylor result on the console.

Please put all your code in a single file called *Q1.cpp*. See the example run below.

Sample run:
Enter x: 5 <enter>
Enter n: 3 <enter>
ExpTaylor result: 18.5
exp(x) result: 148.413
Minimum n for a difference of at most 0.01: 17

**Please display at least 3 fractional digits (for non-zero fractional parts).*

IMPORTANT NOTE: YOUR CODE WILL GO THROUGH AN AUTOGRADER SO MAKE SURE YOUR CODE TAKES THE INPUTS IN THE SAME ORDER AS THE ABOVE EXAMPLE AND ITS OUTPUTS ARE IN THE SAME FORMAT (SPACES, MESSAGES, COLONS etc.).

Q2. *Overloading functions.* [35 points]
You are given a Q2.h file with overloaded function prototypes for *isPalindrome*. Implement this overloaded function into a file named Q2.cpp.
Q2.cpp should only include your function implementation, the necessary #include directives if needed, and should not contain anything else such as the main function or global variable declarations. Test your code using a separate main.cpp file where you implement a sufficient number of test cases. **Submit only your function implementation in Q2.cpp.** You should not change the .h file and should not submit your test code (main.cpp).

IMPORTANT NOTE: YOUR CODE WILL GO THROUGH AN AUTOGRADER SO MAKE SURE YOUR CODE WORKS WITH THE PROVIDED Q2.h WITHOUT ANY CHANGES TO THE .h FILE.

Q3. *Advanced function features.* [35 points]
Implement a *guessSqrt* function (`double guessSqrt(double num);`), which estimates the square root of a number *num* by computing the following formula repeatedly:

nextGuess = (lastGuess + (num / lastGuess)) / 2

The initial value for lastGuess can be any positive value. During computation, if the absolute difference between nextGuess and lastGuess is less than **0.001**, you can stop the iterations and return the result. Otherwise, nextGuess becomes the lastGuess, and the computation continues.

Write a program that evaluates the accuracy of the *GuessSqrt* function you implemented. Your program should ask the user to enter a positive number, call *GuessSqrt* to estimate the square root, and compare the estimate with the correct square root. Your program should display the absolute difference between the correct square root and your function's result on the screen. Please put all your code in *Q3.cpp*.

Sample run:
Enter a positive number: 7.89 <enter>
The absolute difference between the correct square root and the function's result is 2.51945e-10

Note: (1) You can store and print the absolute difference using the double type. Printing a double with cout results in scientific number notation (e.g., 2.51945e-10 above). (2) The result above was generated with

an initial lastGuess value of 1. You can use any positive number you like; the exact difference will be different depending on the initialization, but all results will be <0.001.

IMPORTANT NOTE: YOUR CODE WILL GO THROUGH AN AUTOGRADER SO MAKE SURE YOUR CODE TAKES THE INPUTS IN THE SAME ORDER AS THE ABOVE EXAMPLE AND ITS OUTPUT IS IN THE FORMAT "The absolute difference between the correct square root and the function's result is 2.51945e-10" AS IN THE ABOVE EXAMPLE.