CS 3358 Assignment 2

Due: 11:59pm March 6, 2025 (Thursday)

Instructor: Tsz-Chiu Au (chiu.au@txstate.edu)
Doctoral Instructional Assistant: Minhyuk Park (dmz44@txstate.edu)
Grader: Dhruvil Thummar (kwt32@txstate.edu)

In this assignment, you are asked to implement recursive functions. No loop is allowed. Implementations with loops (that is, with "for" or "while") will not be graded and will not get credits.

Please complete the methods or functions in the codes by replacing "// TODO" with your own codes. You are expected to implement/modify the functions with "// TODO" only. You are not allowed to add new codes or new functions to other parts of the .cpp and .h files.

1. (50%) In `hanoi.cpp`, please implement a recursive function named `moveTowerImpl()` to solve the Hanoi Tower problem. An introduction to the problem can be found at https://www.cs.cmu.edu/~cburch/survey/recurse/hanoi.html. Your program should take an integer n as the input, where n is the number of disks on Tower A at the beginning. The disks on Tower A are disk 0, disk 1, … , and disk (n-1), where disk 0 is the smallest disk at the top of Tower A and disk (n-1) is the largest disk at the bottom of Tower A. The objective is to move all disks to Tower B. Your program should print the list of actions to the standard output. For example, when n = 3, the list of actions is:

```
Move disk 0 from A to B
Move disk 1 from A to C
Move disk 0 from B to C
Move disk 2 from A to B
Move disk 0 from C to A
Move disk 1 from C to B
Move disk 0 from A to B
```

2. (35%) In `pow.cpp`, the function `pow()` calculates x^y (i.e., x^y), where x is a floating-point number and y is an integer. `pow()` depends on a recursive function `powWithPosIntExp()`, which computes the exponentiation base on the recursive equation x^y = x*(x^(y-1)), where y is a positive integer. Please implement both `pow()` and `powWithPosIntExp()` in `pow.cpp`.

   Your implementation should handle *all* values of x and y. You need to pay attention to the cases in which x is zero or y is zero or negative. Please refer to https://en.wikipedia.org/wiki/Exponentiation for the precise definition of the exponentiation. In summary, when x is 0.0 and y is 0, the result is 1.0. When x is 0.0 and y is negative, the exponentiation is undefined, and `pow()` should throw an exception. When x is not 0.0 and y is 0, the result is 1.0. When x is not 0.0 and y is positive, `pow()` uses `powWithPosIntExp()`

to calculate x^y. When x is not 0.0 and y is negative, we rely on x^y = 1/(x^(-y)) to compute the exponentiation; that is, `pow()` uses `powWithPosIntExp()` to calculate x^(-y) and then return 1/(x^(-y)).

3. (15%) Please devise a faster method to calculate `x^y` and implement the method in the recursive function `improvedPow()` and `improvedPowWithPosIntExp()` in `improvedPow.cpp`.
   Hint: Instead of 2^10 = 2 * 2^9, we can alternatively decompose 2^10 as 2^10 = (2^5) * (2^5) = (2^5)^2. When y = 17, which is an odd number, we have 2^17 = 2 * (2^8) * (2^8) = 2 * ((2^8)^2). In both cases, you will not want to repeat the calculation of 2^5 or 2^8. Hence, instead of writing `f(n/2)*f(n/2)`, you should write somethings like `temp = f(n/2); result = temp*temp`. You should deal with the negative y case.

   Not for grading, but for your better understanding of the class, please think about the following two questions:

   1. What are the time complexity of `pow()` and `improvedPow()` in the big-O notation?
   2. To see if you indeed get an "improved" implementation for the x^y calculation, you can try some large (but not too large) `y`, with some small `x` in order to make the final result not beyond the scope of double to be `inf`. For example, try `x = 1.00001` and y = `999999`. `pow()` in `pow.cpp` will crash with an error message such as "`Segmentation fault`" or "`interrupted by signal 11:SIGSEGV`" due to the runtime stack overflow (please recall what it is and why it happens), while `improvedPow()` should successfully return the correct result, which is `22025.1`.

Compilation:

You can compile the programs by the following commands on the departmental servers:

```
g++ -o hanoi hanoi.cpp
g++ -o pow pow.cpp
g++ -o improvedPow improvedPow.cpp
```

Submission:

You should put your implementation of `hanoi.cpp`, `pow.cpp`, `improvedPow.cpp` in a folder named `a2_yourNetID`, and then compress the folder into a zip file named `a2_yourNetID.zip`. For example, if your net id is `zz567`, the folder is `a2_zz567`, and the zip file is `a2_zz567.zip`. Finally, please upload the zip file to Canvas. Please do not put other files in the folder and the zip file.