CS 3358 Assignment 2

Due: 11:55pm Monday, March 11, 2019

Instructor: Kecheng Yang (yangk@txstate.edu)
Instructional Assistant: Cody Blakeney (cjb92@txstate.edu)

In this assignment, you are asked to implement three **recursive** functions, namely `moveTower()` in `hanoi.cpp`, `pow()` in `pow.cpp`, and `improvedPow()` in `improvedPow.cpp`.
Note that, in this assignment, no loop is allowed. Implementations with loops (that is, with "for" or "while") will not be graded and will not get credits.
You are generally expected to implement/modify these three designated functions **only** (except following particular instructions in the .cpp files to uncomment or copy some code), and you are **not** expected you add additional helper functions to implement them.

1. (50') In hanoi.cpp, implement the recursive function `moveTower()` to solve the Hanoi Tower problem (https://www.cs.cmu.edu/~cburch/survey/recurse/hanoi.html).
   Please note that we index disks from 0, i.e., an initial tower of 6 disks contains disks 0,1,2,3,4,5. You should just simply use a cout statement to print a line to indicate the movement of a single disk. The final output (i.e. printed-out on your screen) should be a sequence of such movements, which solve the problem of Hanoi Tower.
   Example Input:
   ```
   3
   ```
   Example Output:
   ```
   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, implement the recursive function pow() to calculate x^y (i.e., x^y). In this implementation, simply use the observation in class slides, x^y = x * x^(y-1). For example, 2^10 = 2 * 2^9. As you can see in the main(), I have already handled the cases "x==0" for you, so you do not need to consider this case in your implementation of pow(); however, you do need to think about all cases of y, including negative integers. So, the code is going to be more than what you have seen in the slides.
   <u>**Hint**</u>: Hopefully, you already knew that x^y = 1/(x^(-y)), e.g., 2^(-2) = 1/(2^2) = 1/4.
   That is, if y < 0 (so that –y > 0), you just need a first recursion step to calculate x^(-y) and return 1/(x^(-y). The calculation of x^(-y) can then fit in the positive "y" case in next recursions

3. (15') In `improvedPow.cpp`, implement the recursive function `improvedPow()` to calculate x^y as well but having better running time.
   <u>**Hint:**</u> you should deal with the negative y case in the same way as the Hint for `pow()`. Then,

think about the observation: instead of 2^10 = 2 * 2^9, we can alternatively decompose 2^10 as 2^10 = (2^5) * (2^5). For odd number of y, for example, 2^17 = 2 * (2^8) * (2^8). In either case, you will not want to do the same "calculation" of 2^5 (or 2^8) twice. That is, using a temporal variable, say temp = 2^5, and then calculate 2^10 as temp*temp, is a more efficient than do the recursive function call twice to calculate 2^5 twice.

<u>Not for grading, but for your better understanding of the class, you should try and think about the following.</u>
Following the comments in improvedPow.cpp, you should be able to compare the running time of `pow()` and `improvedPow()`.
What are the time complexity of `pow()` and `improvedPow()` in big-Oh notation?

Submission:

You should submit your work via the assignment tag in the TRACS system.
You should pack `hanoi.cpp, pow.cpp, improvedPow.cpp` into a single .zip file to upload to TRACS. The .zip file should be named as a2_yourNetID.zip, such as a2_zz567.zip