Create **pointer-based** Stack & Queue functions that can hold both a char and a float in each node, write a “main” function that evaluates an infix expression, using the algorithms in the slides on my website with your stack & queue implementations. The input is terminated by an “^” character. All incoming numbers will be single-digit integers (as characters). You will have to convert the characters to floats and put them in your stack and/or queue. The input rules allow for at least 1, but possibly no spaces or multiple spaces between digits and operators. In the following steps, “print” means, display it on standard output.

1. Read a sequence of characters from standard input (could be console or a piped file or even output from another program) into a queue.
   1. Print the characters in the queue (so we know you got it all). When this is performed, the leftmost symbol printed will be the symbol that is at the “front” of the queue.
   2. Treat the first character inserted into the queue as the 1st character to be extracted in the next step.
2. Use the queue to “feed” the stacking operations to convert the sequence to postfix notation (algorithm 1).
   1. Print the characters AND their values in the entire stack (one line pair per like) starting with the top of the stack after **each** “push” operation. After printing the stack, print a line of === signs (about 10 of them.) Do not disturb the stack when printing, because you will need it in the next step.
3. Print the final element that is on the top of the stack, which will be the answer to the evaluation. Label it as the answer (e.g.: “the answer is: 42”)

Rules:

Be sure to FREE list nodes that are removed from your stack and/or queue

Operands are ONLY the usual: plus, minus, times, divide AND parentheses. Your evaluator should properly handle expressions like: ((2+3)\*4)\*(5+2)\*(3+4)^

**Extra credit (10 points):**  DO NOT ATTEMPT THIS until you KNOW your regular operation is working. Make a COPY of your existing solution naming it with –EXTRA after its name, THEN work on the copy. Allow for \*\* meaning exponentiation (2\*\*3 means 2 cubed). This means you would have to be prepared to “peek” at the last entry of your queue \* to see if it is doubled to form the \*\* operator. If you detect that you now have 2 \* in a row, replace the one on the stack with a special operator, such a $ and add it to your operator priority list as higher than \* or /. Be prepared to handle something like (2\*3)\*\*4 YOU will have to tell your CA to use the exponent test instead of the regular test.

Submit the program as usual.