CSCI 115 Lab 5
DUE: 3/8/15 (Sunday) at 11:59 P.M. on Mulan

For the in-lab portion, you should have a program that reads lines and prints error messages (see below)**. Your in-lab participation score (4 out of the 16 total points) will be based on how accurately your program handles invalid input.** The in-lab portion of the lab will be submitted on **Blackboard**.

For the 8:00 9:50 A.M. lab: the in-lab portion is due **BEFORE** 10:05 A.M. on 3/2/15 (Monday) on Blackboard

For the 10:00 – 11:50 A.M. lab: the in-lab portion is due **BEFORE** 12:05 P.M. on 3/2/15 (Monday) on Blackboard

For this lab, you will be using your linked-list library from Lab 4 (or the solution to it provided on Blackboard) to create a basic calculator that handles arbitrary-sized non-negative integers. Such large integers are stored as linked lists of 4-digit chunks in least-to-most-significant order. The calculator has twenty-six variables (or registers), each named with a single lowercase letter, and supports the following operations on arbitrary-sized integers:

Addition
Subtraction (truncated; see below)
Multiplication
Exponentiation by an int (by repeated squaring; see below)

Input to the program should consist of assignment statements, one per line, each of which should have one of the following formats:

<var> = <digits>
<var> = <var or digits> + <var or digits>
<var> = <var or digits> - <var or digits>
<var> = <var or digits> * <var or digits>
<var> = <var or digits> ^ <int>
  whitespace allowed between tokens or at the end of line

Output should consist of the value being assigned to the variable, followed, in the format (a,f), the number of active and free nodes after the operation has completed.

Note that, of course, you have to take in the input as type string. For example:

string a;

a = “12345678+12345678”;

So you will need to parse the string to distinguish the operator from the operands, and also to know which error message to trigger when appropriate.

1. Error messages in the case of bad input
   variable expected (i.e., the input “9ab” would trigger this error message)
   '=' expected (i.e., the input “a9b” would trigger this error message)
   variable or number expected (i.e., the input “a=+b” would trigger this error message”)
   operator expected (i.e., the input” a=123 456 + 12” would trigger this error message”)
   integer expected (i.e., the input a=2^b” would trigger this error message”)
   end of line expected (i.e., the input “a=123+456 b” would trigger this error message)

2. Output is value assigned to the variable, and then
   (active, free) as in
   123412341234 (4,7)
3. Variables are single lowercase letters
4. Subtraction is truncated, meaning that if a < b, then
   a - b is 0.
5. Lists should be freed when they are not accessible;
   i.e., when a variable is overwritten or when a number
   constant is no longer needed, as in y = 1234 + x
6. Exponentiation should be implemented using the method
   of repeated squaring

For example:

x^13 is calculated as (x^6)^2 * x, where
x^6 is calculated as (x^3)^2
x^3 is calculated as (x)^2 * x

The integer in an ^ (for exponentiation) should be read as an integer, e.g.,:
   int a; cin >> a;

x = 345 678 + 123 (error – an operator is expected between 345 and 678)
x=345678+123 (ok)
x =345678+123 (ok)
x = 345678 + 123 (ok – you can have any number of spaces between an operand and its operator)

Numbers are stored as lists of 4-digit unsigned int's,
least significant at the head of the list
so, the number 345678 is stored as

5678 -> 34 -> //

x = 12345678912345678 (stored as 5678 -> 1234 -> 6789 -> 2345 -> 1)
x = 123 (x is overwritten, so free the old list)

So now the nodes 5678 -> 1234 -> 6789 -> 2345 -> 1 are moved from the linked-list to the free list

And the node 123 is in the linked-list

y = 12345678 + x (should free number 12345678 after adding)