Computer Science Competition
District 1 2016
Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Madison
Problem 2	Nicholas
Problem 3	Oksana
Problem 4	Patricio
Problem 5	Rishabh
Problem 6	Susan
Problem 7	Tomas
Problem 8	Violeta
Problem 9	Walter
Problem 10	Ximena
Problem 11	Yash
Problem 12	Zhenya

1. Madison

Program Name: Madison.java **Input File: madison.dat**

Madison always tells the truth, well, sometimes. She uses the words "and", "or", "not", and "xor" when talking about two things at the same time. For example she would say, "I took out the trash and fed the dog". If indeed she did both things, then she would be telling the truth. However, if she really did not feed the dog, then she would not be telling the truth.

Now if she said, "I took out the trash or I fed the dog.", she would be telling the truth overall if she did indeed feed the dog, but did not take out the trash. As long as at least one of those two situations was true, she would be telling the truth.

What's really confusing is when she says, "xor". This means that she is telling the truth when only one of the two things is true. For example, if she says, "I fed the dog xor I cleaned my room.", she would only be telling the truth if she only did one of those two things. If she did both, it would be a lie, or if she did neither, that would also be false.

We'll represent two or three situations using the letters A, B and C, in some kind of combination expression using NOT(!), AND(*), XOR(^) and OR(+), in that order of priority. In other words, NOT has top priority, and OR has lowest priority. You'll also know the true or false state of each letter.

For example, for the expression A+B, and the values 11, which means A is 1 and B is 1, the final result will be true, because true OR true is true.

For the expression A*B, and the values 10, the final result would be false, since true AND false is false.

There may be parentheses involved as well, in which case the expression inside would be evaluated first. There will be no instances of nested parentheses. It is possible for any of the variables to be used more than once in an expression, like A*B+A*C.

Input: Several lines of data, each line with an expression containing no spaces, followed by a two or three digit string of zeros and ones, representing the corresponding values of the variables in the expression.

Output: The final true or false value of the expression, given the actual values of each variable.

Sample Input:
```
A+B 11
A*B 10
A+B*C 101
A^B 01
!(A+B) 10
```

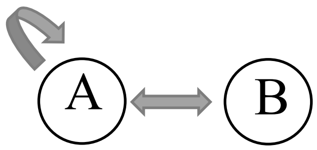
Sample Output:
```
true
false
true
true
false
```

2. Nicholas

Program Name: Nicholas.java **Input File: nicholas.dat**

Nicholas is learning about how to manually count path lengths in a graph, such as the one shown here, using adjacency matrices, but still needs your help to confirm his counting. He started with this simple example, which produces the following adjacency matrix:

```
  A B
A 1 1
B 1 0
```



He has learned that this matrix means that given two vertices A and B in the graph above, there is a connection from A back to itself, and a two-way connection from A to B. To count the number of paths of length one, or direct connections in the graph, all he has to do is count the number of 1s in the graph, three in this case, represented in letter notation as AA, AB, and BA. AA means that the connection starts and ends at A, AB means it starts at A and ends at B, and so on.

However, counting the number of two-hop paths is a little more involved. He can see that AAA is a possibility, and then ABA, and then BAB, but wonders if AA could be combined with B. His teacher says, “Yes, it can.”, so he adds AAB and BAA to his count, making a total of five 2-hop paths.

"What about 3-hop paths?", he wonders. Hmmm, let's see, starting from A there would be AAAA, AAAB, AABA, ABAA, and ABAB. Starting from B, the 3-hop paths are BAAA, BAAB, and BABA. Altogether, that would make eight 3-hop paths within this graph.

Your job is to take any graph with at least two, but no more than five vertices, and help Nicholas confirm his counting of the number of paths of any length from 1 to 3 hops.

Input: Several lines of data, each consisting of an integer N, indicating a graph with N nodes, followed by an integer P, the path length to calculate, followed by N strings, each of length N, containing 0s and 1s, indicating the adjacency matrix for the graph.

Output: The number of P length paths in the given matrix.

Sample Input:
```
2 1 11 10
2 2 11 10
2 3 11 10
3 1 111 001 001
4 2 0100 0010 0001 1000
```

Sample Output:
```
3
5
8
5
4
```

3. Oksana

Program Name: Oksana.java **Input File: None**

Oksana loves to play the Connect Dots game, but gets really tired of drawing out the dots on paper. She wants to be able to print them out, but needs your help.

Below is a picture of an 8X8 grid of dots. The dots in each row have a space between, but the rows do not. Write a program to output this exact pattern.

Input: None

Output: An 8X8 grid of dots as shown below, with a single space following each dot, but no blank lines between rows.

Expected Output:
```
* * * * * * * * 
* * * * * * * * 
* * * * * * * * 
* * * * * * * * 
* * * * * * * * 
* * * * * * * * 
* * * * * * * * 
* * * * * * * * 
```

4. Patricio

Program Name: Patricio.java **Input File: patricio.dat**

Patricio lives in a dynamic world. The grounds on which he lives are magical and will shift a little bit each day. He has built a rocket and wants to know the best time and place from which to launch it. The best place to launch his rocket will be the place of highest elevation. Luckily, Patricio has lived here for a long while, so he has derived a mathematical formula to represent how the grounds will shift in the next few days. These formulas can sometimes be rather complex; he needs your help to determine when and where to launch his rocket.

The formulas are a function of position coordinates on the surface (x, y) and the day, represented by t. These formulas that Patricio has derived can contain parentheses, exponents(^), multiplication(*), division(/), addition(+), subtraction(-) and standard order of operation applies. Also, as a common shorthand, if there are two variables together without an operation between, multiplication is assumed.

Input: The first integer will represent the number of data sets to follow. The first line will be the expression of elevation derived by Patricio. The next line will be the maximum values for x, y, and z, respectively. Single spaces will separate all operands and operators.

Output: The sentence **"Patricio should launch at x=**<*the x coordinate of the highest elevation*> **and y=**<*the y coordinate of the highest elevation*> **at t=**<*the day Patricio should launch*> **at a height of** <*the maximum elevation at which Patricio can launch*>**."** The x and y coordinate should be rounded to one decimal place, and t should be rounded to the nearest day (integer).

Assumptions: All division operations in Patricio's formulas are floating point division; there is no integer division. Division by 0 is considered invalid, and is not considered infinity. The variables x, y, and t can be zero, but they cannot be negative. All formulas Patricio derives are dependent on x, y, and t.

Sample Input:
```
3
z = x^3 + 50x + t^2 + x^2y + x^y + xyt
5.0 5.0 10
z = x^2 + y^2 + t^2
10.0 10.0 100
z = -5x^2 + 4x + -5y^2 + 4y + -5t^2 + 4t
2.0 2.0 10
```

Sample Output:
```
Patricio should launch at x=5.0 and y=5.0 at t=10 at a height of 3975.0.
Patricio should launch at x=10.0 and y=10.0 at t=100 at a height of 10200.0.
Patricio should launch at x=0.4 and y=0.4 at t=0 at a height of 1.6.
```

5. Rishabh

Program Name: Rishabh.java   Input File: rishabh.dat

In researching the concept of string parity, Rishabh encountered some work done by Richard Hamming in 1950. His work produced an innovative way to improve this error-checking process to ensure data is accurately and reliably transmitted. Parity is often referred to as ODD or EVEN, by adding up the bit values in a string. If EVEN parity is used, the string 10110101, which has 5 bits, is appended with the value 1 to make the total number of bits even. If ODD parity is used, the same string would be appended with a 0, to maintain an odd total for all of the bits, resulting in 101101010 as the transmitted string.

Hamming's work produced not only a way to check IF there is an error, but WHERE the error is in the string. Supposedly, a parity code based on Hamming's technique is said to be a perfect code in reliably transmitting and fixing strings that become corrupted. Here is an example using even parity. For a bit string of length 8, which is 2^3, 4 bits can be added to create Hamming parity string. For a string of 2^N bits, the leftmost bit is considered position 1 and the rightmost bit is position 2^N. The parity bits will be placed at each position that is a power of 2 (i.e., at positions 1, 2, 4, 8, 16, …), with the actual data bits starting in position 3, then 5, 6, 7, 9, 10, ... and so on. For the string 10110101, which can be represented by the hex string B5, the Hammond parity string would start out as:

```
1 2 3 4 5 6 7 8 9 10 11 12
_ _ 1 _ 0 1 1 _ 0  1  0  1
```

To find the bit value in position 1, add up all of the odd position bits, with any blank counting as a zero. This would be 0+1+0+1+0+0, for a total of 2, which is even, therefore no bit needs to be added, making the value of position 1 zero.

```
1 2 3 4 5 6 7 8 9 10 11 12
0 _ 1 _ 0 1 1 _ 0  1  0  1
```

To find the bit value in position 2, start at position 2 and use two bits, then skip 2, use 2, skip 2, and so on resulting in summing the values in positions 2, 3, 6, 7, 10, and 11, for a sum of 0+1+1+1+1+0, or 4, again even parity resulting in zero for position 2.

```
1 2 3 4 5 6 7 8 9 10 11 12
0 0 1 _ 0 1 1 _ 0  1  0  1
```

To find the bit value in position 4, start at position 4 and use 4 bits, then skip 4, use 4, skip 4, and so on resulting in summing the values in positions 4,5,6,7, and 12, for a sum of 0+0+1+1+1, or 3, needing a 1 in position 4 to make it even parity. The value of the 8 bit would use the same pattern as the others: use 8, skip 8, and so on, summing the bits in positions 9, 10, 11, and 12, for a total of 2, even parity and zero for position 8. In general, for a parity bit at position P, where P is a power of 2, the general pattern continues in the same manner: start at position P then use P values, skip P values, and so on…

Here is the final transmitted string. The added parity bits in positions 1, 2, 4, and 8 are: **`0010`**.

```
1 2 3 4 5 6 7 8 9 10 11 12
0 0 1 1 0 1 1 0 0  1  0  1
```

For odd parity, the example above would result in an odd parity bit string of **`1101`**.

```
1 2 3 4 5 6 7 8 9 10 11 12
1 1 1 0 0 1 1 1 0  1  0  1
```

Input: Several strings in hex form, each followed by the word EVEN or ODD, to indicate the parity to use. It is guaranteed that the hex string will produce no more than 128 bits.
Output: The resulting Hammond parity string for the given hex string.

Sample Input:

```
B5 EVEN
B5 ODD
CF EVEN
F EVEN
ABC ODD
```

Sample Output:

```
0010
1101
0100
111
11101
```

6. Susan

Program Name: Susan.java Input File: susan.dat

Susan is experimenting with simple encoding techniques to send messages. So far, she can only get single words or word fragments, so her codes don't quite make sense, but she will persist.

The encoded word or fragment is in the form of a sentence, with the last token of the sentence (enclosed in square brackets [ ]) as the regular expression code that divides up the sentence, with an attached adjacent integer indicating the position in the resulting division of the encoded word or fragment. A regular expression enclosed in [ ] means that any of the characters, including a space, are used to divide the sentence. The [AN] example below means that the sentence is split at any occurrence of either the 'A' or the 'N'. If there is a '+', that means any combination of any of those letters will be used as a single division of the sentence.

For example, in the sentence:
`ONE FLEW EAST, ONE FLEW WEST, ONE FLEW OVER THE CUCKOO'S NEST [ ]10`
the output would be `"CUCKOO'S"` since that word is in position 10 of the resulting array.

For the sentence, `ONE IF BY LAND, TWO IF BY SEA [AN]3`
the `[AN]3` divides the sentence into the phrases `"O", "E IF BY L","", "D, TWO IF BY SE "`, with the empty string created by the `"AN"` side by side in the word `"LAND"`.

For, `"METHINKS HE DOTH PROTEST TOO MUCH [O+]2"`, the `O+` divides the sentence into `"METHINKS HE D", "TH PR", "TEST T", and " MUCH "`, since the `"O+"` in the expression means, "one or more Os".

Input: Several sentences, each with a code token and integer at the end.

Output: The resulting word or word fragment for each input sentence and code token.

Sample Input:
```
ONE FLEW EAST, ONE FLEW WEST, ONE FLEW OVER THE CUCKOO'S NEST [ ]10
ONE IF BY LAND, TWO IF BY SEA [AN]3
METHINKS HE DOTH PROTEST TOO MUCH [O+]2
```

Sample Output:
```
CUCKOO'S
D, TWO IF BY SE
TEST T
```

7. Tomas

Program Name: Tomas.java Input File: tomas.dat

In doing some unusual scientific research lately, Tomas has discovered a peculiar sequence in various naturally occurring situations, where certain measurements of growth produce the consistent pattern 1, 1, 1, 3, 5, 9, 17, and so on. He has recorded the data in stages, where stages 1, 2, and 3 are all the value 1, and then stage 4 begins showing observable growth with the value 3 (sum of the previous three values), stage 5 the value 5 (1+1+3), and so on, following the same arithmetic addition sequence. He wants to be able to predict any particular stage, and needs your help. For example, he wants to know what the value would be at up to stage 50 of this unusual growth pattern.

Input: Several integer values N (1 <= N <= 50), each representing a stage of growth according to the pattern described above, each on one line.

Output: The growth size achieved at stage N in the growth pattern described above.

Sample Input:
```
1
3
5
7
9
```

Sample Output:
```
1
1
5
17
57
```

8. Violeta

Program Name: Violeta.java **Input File: violeta.dat**

Violeta loves palindromes. A palindrome is a word that is the same forwards and backwards. As her fascination with them has continued, she has started noticing them more and more often. She has actually started believing that palindromes are everywhere. She has recruited you to write a program to identify the longest palindrome you can find in any given string so she can see if palindromes are indeed as common as she believes them to be.

Input: The first integer will represent the number of data sets to follow. Each data set will contain one line (with no spaces) of the word Violeta wants you to search.

Output: Each data set should have an output that is the longest palindrome found in the input string.

Assumptions: It's a big world out there, and the string Violeta gives you may not be small, so your palindrome finding program may have to be efficient to finish in a reasonable amount of time. It is guaranteed that there will always be a unique "longest length" palindrome, and that each output will be unique and be at least one character in length.

Sample Input:
```
3
abkjlkdfja123456789987654321fewrwefdsfds
beautyandthebeastisagoodfilmthequickbrownfoxjumpedoverthelazydoggodyzalehtrevodepmujxo
fnworbkciuqehtthicketofthebeast (this line is a continuation of the 2nd data set)
racardriversarereallycool
```

Sample Output:
```
123456789987654321
thequickbrownfoxjumpedoverthelazydoggodyzalehtrevodepmujxofnworbkciuqeht
racar
```

9. Walter

Program Name: Walter.java **Input File: walter.dat**

Walter works at a mail center and must decide how to mail certain items brought in by customers, according to the dimensions of the item, from small post cards to large packages. Some items don't fit any of the categories and are unmailable. If an item somehow fits into two categories, the larger one is to be used.

Below are the category descriptions that Walter can choose from. All given dimension ranges are inclusive and expressed in inches.

SMALL POST CARD: Length must be between 3.5 and 4.25, width between 3.5 and 6, and thickness between .007 and .016.
LARGE POST CARD: Length between 4.25 and 6, width between 6 and 11.5, thickness between .007 and .016.
SMALL ENVELOPE: Length - 3.5 to 6.125, width - 5 to 11.5, thickness - .016 to .25.
LARGE ENVELOPE: Length - 6.125 to 24, width - 11 to 18, thickness - .25 to .5.
SMALL PACKAGE: Item dimensions must exceed all rules for large envelope and the length plus the distance around the sides of the package other than the length equals 84 inches or less.
LARGE PACKAGE: Length plus distance around the other sides of the package exceeds 84 inches but is no more than 130 inches.

Input: Several data sets, each on one line, consisting of the three values of a mail item: length, width, thickness.

Output: The correct mail classification (in all caps) according to the specifications listed above, or the word UNMAILABLE if the item does not fit within any of the descriptions.

Sample Input:
```
4 4 .009
5 7 .013
5 7 .2
10 12 .4
20 20 40
```

Sample Output:
```
SMALL POST CARD
LARGE POST CARD
SMALL ENVELOPE
LARGE ENVELOPE
UNMAILABLE
```

10. Ximena

Program Name: Ximena.java **Input File: ximena.dat**

Like Violeta, Ximena loves palindromes, and especially likes to create her own. She'll take a sentence and make a palindrome sentence, taking each word of the sentence, creating a double palindrome for each word. A double palindrome is created when you take the first half of a word, spell it forwards, and then backwards, and then the second half of the word, spell it forwards and backwards, and then connect those two into one long word.

An even more fun thing Ximena loves to do is to take words with an odd number of characters, and do the first half of the word in forwards/backwards order, and then the second part of the word in backwards/forwards order. The middle letter of an odd length word is only used in the second half of the word.

For words of even length, she does the opposite - first half backwards/forwards, last half forwards/backwards.

For example, using her name, XIMENA, she would make a double palindrome that looks like this: MIXXIMENAANE
Using her friend's name, VIOLETA, she would create this: VIOOIVATELLETA
For her other friend, ABE, the result is AAEBBE

Input: A list of capitalized names, each at least two letters and no more than 20 in length.

Output: A double palindrome, as described and demonstrated above, each on a separate line, in all caps.

Sample Input:
```
XIMENA
VIOLETA
ABE
```

Sample Output:
```
MIXXIMENAANE
VIOOIVATELLETA
AAEBBE
```

11. Yash

Program Name: Yash.java　　　　Input File: yash.dat

Yash has just learned in computer science class about **Order of Magnitude,** or **Big O,** and wants to work out the numbers. For example, he has learned that any algorithm with an efficiency of O(1) generally takes 1 step to complete, regardless of the size of the data, with varying larger values for the other levels of efficiency, such as O(log N), O(N), O(NlogN), and O(N^2). He learned in class that for a data set of 10 items, these five values are 1, 4, 10, 40, and 100. He was bit confused by O(log N), until his teacher said, *"Think about the exponent for the power of 2 that equals or just exceeds 10."* He thought, *"OK, 2^1 is 2, 2^2 is 4, 2^3 is 8, and 2^4 is 16. That's why 4 is the log base 2, or O(logN), answer for the value 10! I get it! It's just the integer exponent of the number using base 2, that creates a value equal to or just past the number."*

He then tried to work out higher values of N, but started to get confused again, and needs your help.

Input: Several integers N, each on one line, representing the number of elements of data to be processed by an algorithm.

Output: The five values associated with five levels of Big O algorithm efficiency, as described above, for each value N, with values exceeding 999 shown using comma separation, and a single space between each value.

Sample Input:
```
10
50
100
```

Sample Output:
```
1 4 10 40 100
1 6 50 300 2,500
1 7 100 700 10,000
```

12. Zhenya

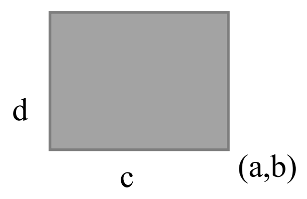
Program Name: Zhenya.java Input File: zhenya.dat

Zhenya works at a construction company that has been stacking clear boxes. From her perch atop the highest view on the crane, she can see the entire landscape of the box stacking. She considers her day finished when all the boxes are stacked. Because the boxes are clear and due to how high up she is though, she can only see the boxes in a 2D plane; she cannot discern the depth or order of the boxes.

This construction work is pretty atypical and boxes are considered to be stacked if they are overlapping at all, or if Zhenya can see one box through the other box. Her work day is finished if from Zhenya's vantage point, looking downward, there is at least one point on the work zone that she can see all the boxes stacked on that point. In other words, she is finished when there is a support rod that can be placed going through all the boxes vertically (through the depth of the boxes).

Please help Zhenya tell if the boxes are all stacked and she is done or if they are not.

Input: The first integer will represent the number of data sets to follow. The line will contain n groups of 4 integers, a, b, c, d. The first 2 integers (a,b) in each group of 4 represents the bottom right point of the box from Zhenya's point of view. The 3rd integer is the width (along the x axis) of the box. The 4th integer is the distance along the y-axis of the box.



Output: For each output, print "ALL STACKED" if there is a point where a rod can be placed through all of the boxes. If the boxes in their current configuration do not have this property, print "NOT STACKED".

Assumptions: Length and width will be positive.

Sample Input:
```
3
0 0 2 2 1 1 2 2 -1 -1 3 3
1 1 300 0 0 0 1 3 10 10 14 44829
0 0 10 10 2 2 5 5 4 4 1 1
```

Sample Output:
```
ALL STACKED
NOT STACKED
ALL STACKED
```