Computer Science Competition
Region 2018
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	Ariana
Problem 2	Bingwen
Problem 3	Caleb
Problem 4	Debra
Problem 5	Emily
Problem 6	Joe
Problem 7	Opus
Problem 8	Prashant
Problem 9	Ramya
Problem 10	Samantha
Problem 11	Vincent
Problem 12	William

1. Ariana

Program Name: Ariana.java **Input File: ariana.dat**

Ariana is an avid Harry Potter fan and recently read about the Black Punishment Quill that Dolores Umbridge invented and inflicted on Harry several times while she served as the Hogwarts High Inquisitor in the book, **Order of the Phoenix**. According to the *Harry Potter Lexicon*, the quill is "**A long, thin black quill with an unusually sharp point, this quill does not use ink. Instead, when used to write, it magically slices into the back of the hand holding it and draws blood in place of ink. The cuts at first heal immediately, but over a long session the damage will be repaired more and more slowly, until the hand is finally left raw and bleeding.**"

The sentence she always instructed Harry to write, over and over again, until his hand was bleeding and eventually scarred permanently, was, "*I must not tell lies.*"

In one gruesome scene of the story which eventually was cut from the book, Professor Umbridge was particularly angry with Harry, and insisted that he not only write the sentence multiple times, but also number each sentence in sequence, and then uppercase the words in sequence, lowercasing the rest. She figured this would affect more skin and cause more pain and suffering on Harry's part, much to her morbid delight.

Write a program that outputs this sentence **40 times**, numbering each sentence, but also uppercases each word in the sentence, one at a time, in forward sequence, and then in reverse sequence once the final word of the sentence is reached, over and over again, like this:

```
1  I must not tell lies.
2  i MUST not tell lies.
3  i must NOT tell lies.
4  i must not TELL lies.
5  i must not tell LIES.
6  i must not TELL lies.
7  i must NOT tell lies.
8  i MUST not tell lies.
9  I must not tell lies.
10 i MUST not tell lies.
11 i must NOT tell lies.
12 i must not TELL lies.
```

… and so on

Input: None.

Output: The list of forty sentences as described and partially shown above, using the pattern indicated. All sentences must be numbered, and then **aligned exactly as shown**.

2. Bingwen

Program Name: Bingwen.java **Input File: bingwen.dat**

Bingwen is calculating third roots of values, has trouble doing it by hand, and doesn't have a scientific calculator with that function, which would certainly make it easy to do. For example, the third root of 8 is 2, which simply means 2x2x2 equals 8. Can you help him?

Input: Several values, each on one line, each no larger than 2147483647, or no smaller than its complement.

Output: The third root of each value, rounded to two decimal places.

Sample input:

```
8
234
6124
-699
84975
-33402
```

Sample output:

```
2.00
6.16
18.30
-8.87
43.96
-32.21
```

3. Caleb

Program Name: Caleb.java Input File: caleb.dat

Caleb's infantry company has been deployed to Bogswamistan for the next two years. Upon arrival Caleb discovers that Bogswamians tell time differently than we do here in the US. Rather than seconds, minutes and hours they simply use degrees as a unit of time. Where once around the face of a clock for us is 60 minutes or 12 hours, once around the face of a Bogswamistanian clock is 360 degrees. Bogswamians simply report the time in two measures of degrees, separated by a slash. For example, in Bogswamian time 12:00 noon for us would be 0/0. 6:00 pm (and am) would be 180/0. 3:45 our time would be 112/270 in their time. Help Caleb write a program that will convert a Bogswamian time to one the rest of his company can understand.

Input: A whole number N representing the number of Bogswamian times to convert to standard time followed by N "degree" times shown as two whole numbers H and M separated by a colon. H and M will both be greater than or equal to 0 and less than 360. None of the Bogswamian times will contain leading zeroes. There will not be a designation for am or pm in either format.

Output: For each of the N Bogswamian times, output a standard US time in hours and minutes. All minute values should be displayed using two digits. Single digit hours should be displayed as a single digit. (Examples: 9:05 10:09 12:15 4:00) A minute in standard time does not elapse until the equivalent degrees in Bogswamian time has elapsed.

Sample input:

```
5
0/0
180/0
112/270
215/60
357/330
```

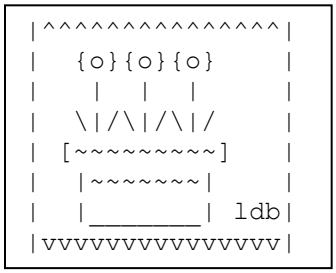
Sample output:

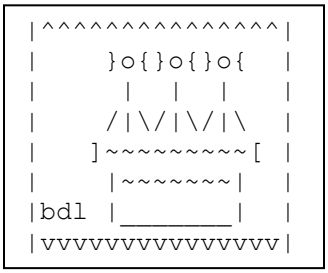
```
12:00
6:00
3:45
7:10
11:55
```

4. Debra

Program Name: Debra.java Input File: debra.dat

Debra loves ASCII art, and has found a really cool website (http://www.chris.com/ascii) that has all kinds of beautiful ASCII graphic pictures, created by using only the characters found on the keyboard. She particularly liked the flowers, and found a cute one she really liked, shown below:





She noticed the initials ***ldb*** as a part of the picture and after looking through the website realized those belonged to the artist who created the picture. Debra decided to play around with the graphic and wanted to see what it would be like if she reversed it. It wasn't quite the mirror image, but pretty close, and it was certainly a fun exercise. Help her by writing a program to output the mirror image of a given graphic, like the example shown above.

Input: Several sets of data, each consisting of two integers R and C, representing the dimensions of a rectangular graphic, enclosed in a frame like the one shown in the example above, followed by R rows with C characters in each row, containing an ASCII graphic picture. All of the blanks are single spaces, no tabs included.

Output: The mirror image of the ASCII graphic, each output followed by "**=====**".

Sample input:

```
8 17
|^^^^^^^^^^^^^^^^|
| {o}{o}{o} |
| | | | |
| \|/\|/\|/ |
| [~~~~~~~~~~] |
| |~~~~~~~| |
| |_______| ldb|
|vvvvvvvvvvvvvvvv|
14 18
|^^^^^^^^^^^^^^^^^^|
|          _.-.    |
|        ,'/ //\   |
|       /// // )  |
|      /// // //|  |
|     /// // ///   |
|    /// // ///    |
|   (`: // ///     |
|    `;`: ///      |
|    / /:`:/       |
|   / /  `'        |
|  / /             |
|(_/  hh           |
|vvvvvvvvvvvvvvvvvv|
```

Sample output:

```
|^^^^^^^^^^^^^^^^|
|     }o{}o{}o{   |
|      |   |   |   |
|     /|\/|\/|\   |
|    ]~~~~~~~~~~[ |
|     |~~~~~~~|   |
|bdl |_______|   |
|vvvvvvvvvvvvvvvv|
=====
|^^^^^^^^^^^^^^^^^^|
|    .-._          |
|   \// /',        |
|  )/ // ///       |
|  |// // ///      |
|   /// // ///     |
|    /// // ///    |
|     /// // :`(   |
|      /// :`;`    |
|       /:`:/ /    |
|        '`  / /   |
|            / /  |
|          hh  /_(|
|vvvvvvvvvvvvvvvvvv|
=====
```

5. Emily

Program Name: Emily.java Input File: emily.dat

Emily has just learned how to manipulate two-dimensional arrays of values, specifically how to search for patterns hidden within such an array. She wants to practice her skills and has come up with a challenge. Taking a square grid of single digits, Emily decides to do a search for numeric strings inside the grid, like the one shown below.

```
1 2 3 4 5 4
2 5 4 3 2 1
3 5 3 4 5 1
4 4 3 4 5 2
5 3 4 3 4 3
1 4 5 2 3 4
```

Being a very methodical and systematic problem solver, Emily always starts at the top left corner of the grid, making her way across that row, then to the next row all the way across, and so on until she reaches the bottom right of the grid. She does this grid traversal for each of the eight directions she is searching, first looking northwards, then towards the northeast, then east, and on around the remaining directions in clockwise order. It is certainly possible that the search string can be found in multiple places, even being in the same row, same column, or same diagonal. It is also possible that the string is nowhere to be found in the grid. Furthermore, Emily decides that the numeric strings can "wrap around" in any of the eight cardinal directions.

For example, the numeric string 3451 is found in two places, first at position (3,3) going east and then at position (3,1) going south. Another string, 425, is found four times: (1,6) SE, (5,3) SE, (6,2) S, and (4,1) W. A "wrap around" example starts at position (6,2), wrapping around to the top of grid, and also starting at position (1,6) and going diagonally towards the southeast, and then going down a row and all the way to the first column for the 2 digit, and the 5 one more row down and one column to the right.

She is able to do small grid searches by hand but needs your help to write a program for processing larger data sets and longer strings, reporting all found instances first by direction (search northwards first, then clockwise from there), then for each search direction going by row, and finally by column.

Input: An initial N, indicating the size of the square grid of single digits, followed by an NxN grid of single digits ranging from 0 to 9, each row on a separate line, with single space separation between digits. Following the grid will be several numeric strings S, also containing the digits 0 through 9, with no spaces between digits.

Output: For each string S, show the string inside angle brackets as shown below, and report the locations where S is found in the grid, in direction, row and then column order, as described above and shown in the sample output below. If the search string is not found, output the words NOT FOUND.

Sample input:

```
6
1 2 3 4 5 4
2 5 4 3 2 1
3 5 3 4 5 1
4 4 3 4 5 2
5 3 4 3 4 3
1 4 5 2 3 4
3451
425
546
5433
351
```

Sample output:

```
<3451>
(3,3) E
(3,1) S
<425>
(1,6) SE
(5,3) SE
(6,2) S
(4,1) W
<546>
NOT FOUND
<5433>
(6,3) N
(5,1) NE
(5,1) SE
(3,2) SW
<351>
(5,4) NE
(3,3) NW
```

6. Joe

Program Name: Joe.java Input File: joe.dat

Joe has discovered a literary classic, a wonderful and poignant story about rabbits, believe it or not, in a quest for survival called **Watership Down**, by Richard Adams, published in 1974. He especially loves the preambles for each chapter, which are quotes from other literary classics. For example, the one for the first chapter is an excerpt from Aeschylus, Agamemnon, where Cassandra and the chorus have an exchange.

Chorus: Why do you cry out thus, unless at some vision of horror?
Cassandra: "The house reeks of death and dripping blood."
Chorus: How so? Tis but the odor of the altar sacrifice.
Cassandra: "The stench is like a breath from the tomb."

These preambles are meant to introduce a particularly strong theme for that chapter, in this case introducing Fiver, a small and troubled rabbit who has visions of impending doom for their home, much like the visions of Cassandra.

Joe decides to use some of these preambles in a simple text processing exercise where he takes all the words of the sentence, uppercases and alphabetizes them by length, eliminating punctuation marks and all duplicates. For example, the result of the lines above would be:

AT DO OF CRY OUT WHY YOU SOME THUS HORROR UNLESS VISION
OF AND THE BLOOD DEATH HOUSE REEKS DRIPPING
OF SO BUT HOW THE TIS ODOR ALTAR SACRIFICE
A IS THE FROM LIKE TOMB BREATH STENCH

Input: Several lines of text as shown in the sample input below. No punctuation symbols are to be included in the resulting words, including periods, commas, question marks, exclamation marks, or quotes. There will be no hyphenated words or contractions in the data.

Output: For each line of text, output all words in alpha order by size, as described above and shown below.

Sample input:

```
Why do you cry out thus, unless at some vision of horror?
"The house reeks of death and dripping blood."
How so? Tis but the odor of the altar sacrifice.
"The stench is like a breath from the tomb."
The darksome statesman, hung with weights and woe
Like a thick midnight fog, moved there so slow
```

Sample output:

```
AT DO OF CRY OUT WHY YOU SOME THUS HORROR UNLESS VISION
OF AND THE BLOOD DEATH HOUSE REEKS DRIPPING
OF SO BUT HOW THE TIS ODOR ALTAR SACRIFICE
A IS THE FROM LIKE TOMB BREATH STENCH
AND THE WOE HUNG WITH WEIGHTS DARKSOME STATESMAN
A SO FOG LIKE SLOW MOVED THERE THICK MIDNIGHT
```

7. Opus

Program Name: Opus.java Input File: opus.dat

The Firethorne High School Computer Science Club is sponsoring an invitational computer science competition to raise money to pay for much needed compute time on The University of Texas' super computer Stampede. The invitational will be just like all other UIL competitions in that there will be a programming portion and a written exam. The club sponsor, Mr. Snigglefritz, has asked Opus to help out with grading the written portion of the contest. Each contestants name, name of their school, school's classification (1A, 2A, etc.) and their answers to the 40 written exam questions will be scanned into a file along with the key to the exam. Opus must write a program that will read and score each contestant's answers. It must then print a report that separates the contestants by school classification and ranks them based on their score on the test.

The usual UIL scoring methods will be used. Contestants will earn 6 points for a correct answer, lose 2 points for an incorrect answer and 0 points for any question that is skipped. Ties will be broken using percent correct. The student with the greatest percent correct will win the tie.

Input: The first line of the data file will be the word KEY followed by 40 letters A through E that represent the correct answers to the written exam each separated by a space. Next will be several lines of data each containing the following: first name, last name, school name (all will be one word), school classification and 40 capital letters A through E or an S if the question has been skipped. Each of the data items will be separated by a single space.

Output: A report that shows the results of the competition. The report should be grouped by school classification from 1A to 6A. Within each classification the contestants should be listed in descending order based on their test score each on a separate line. For each contestant show their placing, last name followed by a comma, their first name, their school's name, and their score on the exam. All items should be separated by a single space.

Ties should be broken and displayed in the correct order with correct place. The tie breaker goes to the student with a higher percentage of correct answers. If two students have the same score and same percentage of correct answers, the tie is considered to be unbreakable. Unbreakable ties should be shown as having the same place and in alphabetical order based on last name, then first name. The next place after an unbreakable tie should reflect the same place, not the next place. For example, if there is an unbreakable tie for first place, list both contestants as first and then list the next contestant as third place, not second.

Sample input:

```
KEY A E C B C B D C A E A C E B D E C D A C E C C B C D E E A C A B B B D B C D A D
Sebastian Williams Channing 1A A E C B C E D C E E E C E C E E C E A C E C D A B S S B A C A B E A D B C C E S
Alexander Taylor Gorman 1A A A S B A S D B S E C S E C S E D S A D E C B S C A E B B C A B S B B S C B S D
Jose Rodriguez Shamrock 1A S E C B C B D S A E E B C C D E C C A A A S E B B D E C A S S A E D D E S B A E
Angel Harris Rusk 4A A E D D C E B C C E A C A A B E A B B C E E B C D A A C A A D A B B C C E D A A
Sofia King Odessa 6A A E C B C B C C B E A C E C D B S B D S C D S D C E A C E S D D S B E S D D S S
Aria Morris Frost 2A C C C C C C C C C S S S S S S S S S S C C C C C C C C C C C C C C C C C C C C C C
Jacob Miller Zapata 4A A A D E E B B D B B E A B B D E C A E E E C C B A A D C C B C C D E E D C E E B
Michael Martinez Frost 2A A E D B E B D C A E E B D C D E E C A C B C C S C C S E B S D S S S S B C S A S
Emily Adams Alvin 6A A E C B C B D B A E E C E C D E C C A C E C D B C D E C A C A E E B C S S S S S S
Isaac Robinson Alvin 6A C E C A D B B C E E E B E B D E C A A D E A D E C A E C A D A A E E D C C A A A
Dylan Lopez Tulia 3A C A A B E E A B B C E D D B D B C C E D C E B D C B D D A B E E E D A D C B B D
Camila Green Poolville 2A A E C A C B D C A D E C E C D E C C A C E C D B C D S B S S A E E B D B S E A D
Penelope Stewart Van 4A D E C B C B D C C B E C E C D E C D E C E C D E C D E D A C A S E B S B C E A D
David Wilson Rusk 4A A E C B C B D C A E E C E C E E C C A C B D B E S C C E B A S C B E S E C S E C
Scarlett Turner Junction 2A A E D B E B D C A E E B D C D E E C A C B C C S C C S E B S D S S S S S B C S A S
Jayden Jackson Poolville 2A A E C B C B D C A E E C E C D E C C A C E E A D E A D E D D C C D D B C C D B S
Elijah Davis Lasara 2A C E C B C B E C A E E C E C E E C E A C S C D S C D E E A C A B S S C D C B E A
Abigail Baker Lamesa 4A C C E C C B E B A E E C E D E S B C S S S S S S C C C S S S S S S S S S S S S S S
Natalie Flores Marfa 1A A E C B C B D C A E E C E C D S S S S S S S S S S S S A A A A A S S S S S S S S S
Luke White Poolville 2A A E C B C B D A A E E C E C D S S S S S S S S S S S S S S S S S E E E E S S S S S
James Martin Hooks 3A A E C B C D D C C E E C E A D S B S S S E S S S S D E S S S S C E E S S S S S D
```

Opus - continued
Sample output:

```
1A
1 Williams, Sebastian - Channing: 110
2 Flores, Natalie - Marfa: 80
3 Taylor, Alexander - Gorman: 76
4 Rodriguez, Jose - Shamrock: 76
2A
1 Green, Camila - Poolville: 136
2 Davis, Elijah - Lasara: 120
3 Martinez, Michael - Frost: 90
3 Turner, Scarlett - Junction: 90
5 Jackson, Jayden - Poolville: 90
6 White, Luke - Poolville: 58
7 Morris, Aria - Frost: 12
3A
1 Martin, James - Hooks: 74
2 Lopez, Dylan - Tulia: -16
4A
1 Stewart, Penelope - Van: 140
2 Wilson, David - Rusk: 80
3 Harris, Angel - Rusk: 40
4 Baker, Abigail - Lamesa: 16
5 Miller, Jacob - Zapata: 8
6A
1 Adams, Emily - Alvin: 138
2 Robinson, Isaac - Alvin: 72
3 King, Sofia - Odessa: 56
```

8. Prashant

Program Name: Prashant.java Input File: prashant.dat

Prashant has just learned about permutations and knows how to calculate how many possible unique permutations can be generated from a simple list of items. For example, he understands that given three elements A, B and C, there are six different ways these three elements can be listed in order: ABC, ACB, BAC, BCA, CAB, and CBA. A way to express the total is to use the factorial function in mathematics, expressed as 3!, 3 being the number of elements, and the ! operator indicating a "count down and multiply" process, like this:

3 x 2 x 1 = 6

Furthermore he learned that if there are duplicates, like in the set of elements A, B, C and C, there are indeed 4!, or 24 permutations, but since C is duplicated, there are only half as many unique permutations, which would be 12:

```
ABCC, ACBC, ACCB, BACC, BCAC, BCCA, CABC, CACB, CBAC, CBCA, CCAB, CCBA
```

The general expression to reduce the value due to duplicates would be **num!/dup!**, where **num** is the number of elements, and **dup** is the number of duplicates. This formula would still work if the value of **dup** was zero (indicating no duplicates), resulting in 0!, or 1, since by definition, 0! is equal to 1. For the two examples so far, the calculations would be:

3!/0! = 6
4!/2! = 12

However, things got a bit more interesting when he tried the sequence A B A B, which had **two** elements duplicated. He discovered that he needed to alter the denominator of the formula, accounting for the multiple duplicates, resulting in this formula, **num!/(dup1!* dup2!),** which would produce the expression:

4!/(2!*2!) = 6

With even more complex sequences, for each element that was duplicated, he had to add another factor to the denominator to account for each additional duplication. For instance, the sequence A B C A B C D A has three elements that are duplicated, one even occurring three times, requiring a denominator of 3!*2!*2! The resulting number of unique permutations for this list is 630.

Prashant realized this was getting quite complicated and decided to reach out to you for help. He is only interested in calculating the number of **unique permutations** that would be generated, given a list of elements, and not the actual permutations. Can you help him write a program to do this?

Input: Several lines of data, each containing a list of elements represented by single uppercase letters, with single space separation, each sequence possibly containing duplicate elements as demonstrated above. There will be no more than 15 elements in each line of data.

Output: The total number of unique permutations that would be generated with the given data, eliminating any duplicate permutations due to duplicate elements.

Sample input:

```
A B C
A B C C
A B A B
A B C A B C D
```

Sample output:

```
6
12
6
630
```

9. Ramya

Program Name: Ramya.java **Input File: ramya.dat**

Ramya's friend Prashant has been researching permutations, specifically calculating the number of unique possible sequences that can be generated from a list of elements. However, Ramya wants to actually generate the list of unique permutations, which she considers much more of a research challenge than just counting how many there could be, as Prashant is doing.

She knows that given three elements A, B and C, there are six different ways these three elements can be listed in order, starting with the original string, and proceeding from there as follows, ABC, ACB, BAC, BCA, CAB, and CBA, a total of six unique sequences.

She also knows that if there are duplicate elements, as in the list A, B, C and C, there are duplicate sequences, which she wants to eliminate. The unique list would be as shown below.

```
ABCC, ACBC, ACCB, BACC, BCAC, BCCA, CBAC, CBCA, CABC, CACB, CCAB, CCBA
```

From Prashant's research, she also knows that with multiple duplicates in the list, the process becomes even more complicated, and with more and more elements, the list of unique permutations can become quite long. She decides that she will stop generating permutations once the list reaches 30 permutations, even though there could be many more to follow. She needs your help to write the program to generate up to the first 30 unique permutations of a list of elements, in the order pattern demonstrated in all the examples shown on this page, starting with the original sequence.

Input: Several lines of data, each containing a list of elements represented by single uppercase letters, with single space separation, each sequence possibly containing duplicate elements as demonstrated above. There will be no more than 7 elements in each line of data.

Output: The numbered list of unique permutations for each given list, in the exact order pattern and format shown below, each sequence ending with the single line "=====".

Sample input:

```
ABC
ABCC
ABAB
ABCCC
```

Sample output:

```
ABC
ACB
BAC
BCA
CBA
CAB
=====
ABCC
ACBC
ACCB
BACC
BCAC
BCCA
CBAC
CBCA
CABC
CACB
CCAB
CCBA
=====
ABAB
ABBA
AABB
BAAB
BABA
BBAA
=====
ABCCC
ACBCC
ACCBC
ACCCB
BACCC
BCACC
BCCAC
BCCCA
CBACC
CBCAC
CBCCA
CABCC
CACBC
CACCB
CCABC
CCACB
CCBAC
CCBCA
CCCBA
CCCAB
=====
```

10. Samantha

Program Name: Samantha.java Input File: samantha.dat

In accounting class, Samantha is learning how to do simple calculations of subtotal, tax and total, using 8.25% for the tax. She is required to take a list of values, add them up, and output these three values in a column format. For now she will only deal with total values of less than $10,000. Can you help her by writing a program to do this to help her check her work?

For example, a list of values like this:

```
9.99 10.53 45.87 100 613.25
```

would have a final result of this:

```
Subtotal: $ 779.64
Tax     : $  64.32
Total   : $ 843.96
```

Input: Several lines of values, each set of values on one line, with single space separation.

Output: The accounting result of the line of values, as described and shown above, as well as in the samples below. Each complete output is to be ended with "**=====**".

Sample input:

```
9.99 10.53 45.87 100 613.25
12 13 14 15 16 17 18
```

Sample output:

```
Subtotal: $ 779.64
Tax     : $  64.32
Total   : $ 843.96
=====
Subtotal: $ 105.00
Tax     : $   8.66
Total   : $ 113.66
=====
```

11. Vincent

Program Name: Vincent.java Input File: vincent.dat

Vincent was born on April 19, 2002 at 5:45 AM. Recently he has been thinking what the new date and time would be after a certain number of minutes had elapsed since that original time. For example, it was easy for him to think about a relatively small number of minutes, like 60 or 420 or even 1440 minutes later, in essence an hour, seven hours and 24 hours later.

He decided to represent his birth data as follows, and then list the number of elapsed minutes on the next line, like this:

```
2002 4 19 5 45
60 420 1440
```

He also decided on the date/time output format shown below to represent his birth date, followed by the other dates and times he thought about:

```
April 19, 2002 - 05:45 AM
-----
April 19, 2002 - 06:45 AM
April 19, 2002 - 12:45 PM
April 20, 2002 - 05:45 AM
```

He needs your help to write a program that will do this, given any starting date and time, followed by any number of minute values representing number of minutes elapsed from then. **Be sure to account for any leap year situations.**

Input: Several sets of data, each set occupying two lines, the first of which is the original date/time in the format:

yyyy mm dd hh mm

followed by a second line with several integer values, single space separation, representing elapsed minutes since the original time. He didn't consider any year before 1970 in his test data, represented hours in the range 0-23, and minutes from 0-59. The month and day numbers matched the date intended, with months ranging from 1-12, and day numbers ranging according to that month.

Output: The original date formatted as shown in the example above, followed on the next line by "**-----**", and then on the next several lines by the dates and times that represent the elapsed minute values in the second data line, ending with "**=====**" to end the output for that data set.

Sample input:

```
2002 4 19 5 45
60 360 1440 380 1095 4621
1990 8 6 17 39
2526861 81741 2368706 48621
```

Sample output:

```
April 19, 2002 - 05:45 AM
-----
April 19, 2002 - 06:45 AM
April 19, 2002 - 11:45 AM
April 20, 2002 - 05:45 AM
April 19, 2002 - 12:05 PM
April 20, 2002 - 12:00 AM
April 22, 2002 - 10:46 AM
=====
August 06, 1990 - 05:39 PM
-----
May 27, 1995 - 12:00 PM
October 02, 1990 - 12:00 PM
February 06, 1995 - 03:05 PM
September 09, 1990 - 12:00 PM
=====
```

12. William

Program Name: William.java Input File: william.dat

William is fascinated with palindromes, and knows many of the classic ones, like "**Madam, I'm Adam**", "**Rise to vote, sir**.", and "**A man, a plan, a canal, Panama**", but is bored with those and is now looking at a new challenge, palindromes involving sequences of numeral digits, like 6105273940456320451987605431261 4. He clearly sees that this list of digits is not a palindrome but wonders if there is one hiding inside. He decides to divide the sequence into two parts, like this: 6105273940456320 4519876054312614.

He immediately sees a 61 in the first part matched by a 16 in the second part, which would be considered an imbedded palindrome, written like this, "61:16". He decides to look further and see if there was one that was longer, and sure enough, he sees "613:136", and then a longer one, "6134:4136". He keeps searching and finally decides that the largest embedded palindrome he can find is "123450:054321".

For curiosity's sake, he takes the first sequence, chops off one digit at the end, and tries it again. When he splits up the digits this time he gets 610527394045632 0451987605431261, the first list one character shorter than the last. He is surprised that the resulting palindrome - 6157940:0497516 - is actually longer than the first example, which he finds very interesting.

This searching process took a lot of trial and error, and William could use your help in finding a better way to do this, perhaps a computer program that does all the work for him.

Input: Several lines of data, each containing a list of contiguous digits.

Output: The largest "palindrome" contained in the list, printed in the format described above and shown below. If no palindrome exists, output a single colon with no values on either side.

Sample input:

```
6105273940456320451987605431261 4
6105273940456320451987605431261
883625174635
123456789
```

Sample output:

```
123450:054321
6157940:0497516
36:63
:
```