Computer Science Competition
Region 2018
Programming Problem Set

CORRECTED PROBLEM STATEMENTS

The following pages include revised problem statements for the problems listed below. These revised versions contain important corrections and clarifications. Please remove the original pages for these problems from your student packets, and distribute these corrected pages to students at the beginning of the contest – one copy per team.

Number	Name
Problem 3	Caleb
Problem 7	Opus
Problem 11	Vincent
Problem 12	William

3. Caleb

corrected

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 division sign. 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
```

7. Opus

corrected

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 percentage correct **based on number of questions answered**. 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 S C E E S 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
```

11. Vincent

corrected

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
60 420 1440 380 1095 4621
1990 8 6 17 39
2526861 81741 2368706 48621
```

Sample output:

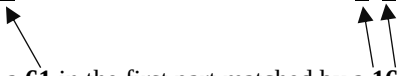
```
April 19, 2002 - 05:45 AM
-----
April 19, 2002 - 06:45 AM
April 19, 2002 - 12:45 AM
April 20, 2002 - 05:45 AM
April 19, 2002 - 12:05 PM
April 20, 2002 - 12:00 AM
April 22, 2002 - 10:46 AM
=====
August 06, 1990 - 05:39 PM
-----
May 27, 1995 - 12:00 PM
October 02, 1990 - 12:00 PM
February 06, 1995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

12. William

corrected

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. 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, **61**0527394045632 0451987605431261, with the first half one digit smaller than the second due to the odd length.



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 "613:316", and then a longer one, "6134:4316". He keeps searching and finally decides that the largest embedded palindrome he can find in the two parts, **61**0**5**2**7**3**940**45632 **04**51**9**8**7**60**5**43**1**2**6**1, is "6157940:0497516".

In another example using the number sequence `883625176435`, which divides evenly into `88`**`36`**`25` and `17`**`6`**`4`**`3`**`5`, he finds **`36`** and **`63`** as the longest embedded palindrome.

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
883625176435
123456789
```

Sample output:

```
6157940:0497516
36:63
:
```