2016/2017 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 1
Dim Weights

Parcel delivery companies generally determine the charge for delivering a parcel based on weight. However, weight is not the only factor that determines what the costs are to handle a given parcel. Space on transport and delivery vehicles is limited. If a parcel is large but light charging only by weight will not accurately reflect the total costs of handling that parcel.

To address this problem, delivery companies calculate a "dim" (for "dimensional") weight. This is determined by taking the total volume of a parcel (length by width by height, in inches) and dividing it by a stated factor to get a value in "pounds," taking any fractional part as the next whole pound. (For example, a result of 3.01 will be treated as 4.) The result is the "dim weight." The charge to ship a parcel is based on the effective weight--the greater of the actual weight or the "dim weight." The "dim weight" divisor varies between delivery companies. Additionally, some delivery companies only apply the "dim weight" to parcels that exceed a certain volume.

Your team is to write a program that will read the three dimensions of rectangular parcels and determine which delivery company has the most favorable treatment for each parcel.

Input to your program will consist of two parts. The first part begins with a line containing the number of delivery companies to be compared, as an integer c beginning in the first column. The next c lines contain the following information about each delivery company:

- the name of the company, as a string of one to twenty alphanumeric characters,
- the minimum parcel volume that the "dim weight" applies to in cubic inches, as an integer between 1 and 9,000 inclusive, and
- the "dim weight" divisor, an integer in the range 10 to 999 inclusive.

Fields on these lines are separated by one or more spaces.

The remainder of the input consists of parcel descriptions, one per line. Each parcel description contains the length, width, and height of the parcel in inches (as an integer in the range 1 to 72 inclusive), followed by the weight in pounds (as an integer in the range 1 to 70 inclusive). Values are separated from each other by one or more spaces. This input is terminated by end-of-file. No input line will exceed 80 characters.

For each parcel, your program is to determine which company will treat the package as having the lightest weight. Print a line containing the name of the company, a single space, and the effective weight as an integer. No leading or trailing whitespace is to appear on an output line.

Should more than one company offer the same lightest effective weight, your program is to select the company that appears first in the company list.

Sample Input

```
4
USEX  1    166
DPS   1    166
Post  1728 192
Local    1     240
8 8 8 3
6 6 6 3
13    9   2   1
13  13  15  5
```

Problem 1
Dim Weights (continued)

Output for the Sample Input

```
Post 3
USEX 3
Post 1
Local 11
```

2016/2017 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 2
Great Thief

Once upon a time, there was a great thief who used the code name FWX. He lived in a two-dimensional world named Flat Land. In this land, houses are built only on the grid locations having integer coordinates, e.g., for a square Flat Land with a highest coordinate of n, there can be houses only at $(0,0)$, $(0,1)$, ..., $(0,n)$, $(1,0)$, $(1,1)$, ..., $(1,n)$, $(2,0)$, $(2,1)$, ..., $(2,n)$, ..., $(n,0)$, $(n,1)$, ..., (n,n). There are only non-negative coordinates in Flat Land.

FWX liked the time between 4 AM to 5 AM for his work when all the inhabitants were sound asleep. In some houses, he slipped and fell with all the things he took in his bag, making a loud sound and knocking himself out. All the inhabitants of the house were awakened and they called the police.

FWX was a great thief, so no jail could keep him for a long time. He broke out of jail and thought about why he was caught only in some particular houses. Then he discovered that there is a guard in the house at $(0,0)$ who can shoot slippery material in a straight line from $(0,0)$ which hits only the first house on that line and makes that house slippery. When the guard fires, say, along the line $x = y$, it makes the house at $(1,1)$ slippery, but houses at $(2,2), (3,3)$, etc. are not affected. Similarly, along the line $2y = 5x$, $(2,5)$ becomes slippery but not $(4,10)$. The guard has an unlimited supply of slippery material and fires in any direction he wishes as often as he likes. See Figure 1.

Your team is to write a program that will find the number of houses which are not safe for the great thief FWX, assuming that the guard has shot slippery material on every possible line. Remember, $(0,0)$ is not safe because of the guard.

Input to your program will be a series of test cases, one per line, terminated by end of file. There will be at most 1000 test cases. A test case is a single integer $n, 1 \leq n \leq 1{,}000{,}000$ which specifies that the Flat Land for this case is a square with the highest coordinate of n.

For each case, print a line containing the number of unsafe houses in that Flat Land on a separate line with no extra spaces or leading zeroes.

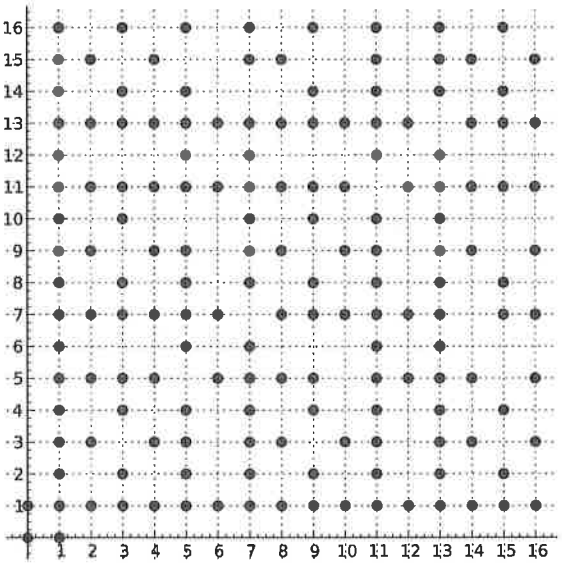

Figure 1. Example Square Flat Land with $n = 16$.

Problem 2
Great Thief (continued)

Sample Input

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
1000000
```

Output for the Sample Input

```
4
6
10
14
22
26
38
46
58
66
86
94
118
130
146
162
607927104786
```

2016/2017 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 3
What's a Cubit?

In ancient Egyptian and Biblical times, length measurements were based on the human body: a digit was the width of a finger, four digits made a palm, and seven palms made a cubit (presumably the length of an outstretched forearm from the elbow to the middle finger).

Egyptian cubit rods have been found, with lengths of 523.5 to 529.2 mm (20.61 to 20.83 inches). These were apparently standardized "royal" cubits. The rods are divided into palms, with the palms subdivided into digits.

Using the mean length of the cubit as 526.35 mm, your team is to write a program that will take ancient dimensions in cubits, palms, and digits, and print the results both in meters and in U. S. feet and inches. A U. S. inch is defined as 25.4 mm, with twelve inches to the foot.

Input to your program will be a series of lines. Each line will contain an ancient measurement specified as integers followed by units, of the form "a cubits b palms c digits". If the value for the number of cubits, palms, or digits is zero, it will be omitted in the input. If the value is one, the unit will be specified in the singular. Fields on an input line are separated by one or more spaces. No input line will exceed 80 columns. Lengths will be greater than zero and less than 1,000 cubits.

For each input line, your program is to print the length in meters (rounded to three digits after the decimal point) followed by the letter 'm'. This is to be followed by a single space and the length in feet and inches. If the length is less than one foot print only the inches; otherwise, print the number of feet as an integer immediately followed by a single quote and a space. The number of inches are then to be printed with two digits after the decimal point (rounded), and immediately followed by a double quote. No leading or trailing whitespace is to appear on an output line.

Sample Input

```
300 cubits
2 cubits 1 palm 2 digits
3 palms 3 digits
1 cubit 2 digits
```

Output for the Sample Input

```
157.905m 518' 0.73"
1.165m 3' 9.89"
0.282m 11.10"
0.564m 1' 10.20"
```

2016/2017 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 4
Robot Scarecrow

A farm deployed a robot as a moving scarecrow for its corn field. The robot moves along a loop pathway that surrounds the corn field. On the pathway there are a number of unmanned command posts, numbered sequentially starting from 1.

Every day the robot starts at post number 1 and moves in the direction of post number 2. Then, for the whole day, every time the robot reaches a command post it receives a command, which can be "continue to the next command post", "report and remain at this command post", or "turn back and return to the previous command post".

Your team is to write a program that will, given the list of commands received by the robot during the day, determine how many times the robot was at a specified command post.

The first line contains three integers N, C and P indicating respectively the number of posts ($2 \leq N \leq 10^3$), the number of commands ($1 \leq C \leq 10^5$) and a specific command post ($1 \leq P \leq N$). Values on this line are separated from each other by one or more spaces. The remaining input consists of C lines, with each line containing a number representing a command: 1 for the command "continue to the next command post", 0 for "report and remain at this command post", or -1 for the command "turn back and return to the previous post". The robot starts at post number 1, turned in the direction of post number 2, and the first command value is always 1, so the robot's first movement is always from post 1 to post 2. A "continue to the next command post" command at post N causes the robot to move to post 1; a "turn back and return to the previous post" command at post 1 causes the robot to move to post N. No leading or trailing whitespace will appear on an input line.

Output a line containing one integer, the number times the robot was at post number P, without leading zeroes or leading or trailing whitespace.

Sample Input

```
10 9 2
1
1
1
1
-1
0
-1
-1
1
```

Output for the Sample Input

```
2
```

2016/2017 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 5
Package Weights

Acme Component Manufacturing creates packages of fasteners for companies that produce assemble-yourself furniture. These packages contain fasteners of various types that are used in the assembly process. The packages are put together by machines that dispense the desired quantities of each fastener.

The machines reliably select the required fastener types for the packages. However, during the packaging process these machines sometimes dispense one too few or one too many of each fastener type. As a check on the process, the resulting packages are weighed. Each type of fastener has an expected weight and a tolerance–one fastener of that type will have a weight within the specified range. (For example, if a nut weighs 0.2 ounces plus-or-minus 0.01 ounce, the weight will be in the range 0.19 to 0.21 ounces inclusive.) The desired number of each fastener is known, therefore the total weight of a correct package can be determined.

However, given the number of fastener types available, it is possible that in some cases the weight might be in range, but the contents might still not be correct, due to possible dispensing errors as described above. Your team is to write a program that will, given a set of fastener types and desired quantities, take package weights and determine if the package:

- definitely contains the proper fasteners,
- definitely does not contain the proper fasteners, or
- the weight is ambiguous—the package might or might not contain the proper fasteners.

Input to your program will consist of several sections. The first section is a list of between 1 and 50 fastener types, one per line, with the following fields separated by commas:

- *Fastener Name:* string of 1–36 alphanumeric characters, possibly including spaces and/or hyphens.
- *Fastener Standard Weight:* floating point value in the range 0.01 to 20000.00 inclusive.
- *Fastener Weight Tolerance:* floating point value in the range 0.0001 to 100.0000 inclusive.

This list will end with an empty line.

The second section contains a list of expected fastener package contents. There will be 1 to 20 fastener packages. The first line for each package contains a package name of 1–36 alphanumeric characters, possibly including spaces and/or hyphens. The remaining 1–14 lines for that package contain a fastener name, a comma, and a fastener quantity. The fastener name will appear in the fastener type list. The same fastener type will not appear more than once in the list for any given package. Package lists will appear one after another in the input. The total expected weight of a package will not exceed 100,000. The list of packages will also end with an empty line.

The remainder of the input will be a list of package names and actual package weights, one per line. The package name will be separated from the actual package weight by a comma. This input ends with the end-of-file. No leading or trailing whitespace will appear on any input line.

For each weighed package your program is to determine whether the package is correct or not—if that can definitely be determined. If the package is definitely correct for the given package name, print a line containing only the word "pass". If the package is definitely not correct, print a line containing only the word "fail". If it cannot be determined if the package is correct or not, print a line containing only the word "ambiguous". No leading or trailing whitespace is to appear on an output line.

Problem 5
Package Weights (continued)

Sample Input

```
Bolt,10.0,0.1
Bracket,17.0,0.2
Mounting Plate,35.0,1.0
Nut,8.0,0.1

Package A
Bolt,4
Nut,4
Bracket,2
Mounting Plate,1
Package B
Bolt,2
Nut,2
Bracket,1

Package A,141
Package A,145.5
Package B,52.87
```

Output for the Sample Input

```
ambiguous
fail
pass
```

2016/2017 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 6
Savage Pokémon

Mysterious parks are being discovered throughout California this year. These parks are unique because there are Pokémon wandering around the park. However, there is a chance that a visitor who enters one of these parks may never come out. Despite this risk, visitors come to these parks in the hope of getting to a location where there is a chance of catching a rare Pokémon never seen before.

These parks are divided into many sections. When visiting such a park, you are randomly assigned a starting section, a destination section, and a series of sections to visit. Some people get trapped and are not able to leave the park. This happens when there is no way to get to their destination section by way of the assigned sections. Some people reach the destination section, but fail to catch the mysterious Pokémon. You want to avoid being trapped because you cannot get to your destination section given your assigned random path, so you compile a list of all the parks, how they are connected and determine which parks to visit using your computer science knowledge.

There is a cylinder at the entrance of the park which brave visitors step into to get teleported to a random start section of the park. The park consists of multiple sections labeled by numbers. A random list of sections you must visit is loaded into your assigned Pokédex. The Pokédex is a digital device that has the information you need about the paths in the park and your assigned start and end section.

For example, a random start section 1 is assigned to you, along with a random list of sections ending at section 6. Example park sections are connected as follows: 0 to 1, 1 to 2, 2 to 3, 2 to 5, 3 to 4, 0 to 6. If you are assigned a random section list of 1, 2, 3, 4, 6, you are trapped because there is no way to reach section 6 from section 4. However, if the park also had a route from 4 to 0, then you could use the route 4 to 0 to 6 to successfully get to the destination. The connectivity maps for the parks are available to the public outside of the parks. The map of the park gets loaded into your Pokédex. Connections are one-way: if a map indicates section 0 connects to section 1, that same connection cannot be used to go from section 1 to section 0.

You need to follow the path from your random start section to a random destination indicated in your Pokédex, visiting each assigned section in turn. Your time in each section is limited and your Pokédex will alert you that it is time to exit each section so that you can move to your next section. Some visitors at some parks cannot get to their randomly generated final destination so they get trapped and risk being eaten by savage Pokémon. You decide to write a program that will use the information about how the sections are connected in each park to determine if you will be able to get from any start section to any destination section no matter what sections you are assigned to visit. If you find a park in which you are not able to follow a path to every other section from any possible start section, you need to avoid it. Otherwise, you will go visit it.

The first line of the input is an integer T, which is the number of the parks you are considering visiting. T sets of park information follow. Each set of information starts with two integers: N, the number of sections in the park in the range $0 < N \leq 100{,}000$, and R, the number of section connections, in the range $0 < R \leq 200{,}000$. The next R lines each contain two integers X and Y, $0 \leq X, Y < N$, meaning that there is a connection from section X to section Y. No sections connect to themselves, nor will there be duplicate connection entries. Each section will have at least one other section that connects to it, or that it connects to.

For each park, your program is to print a line that indicates whether you will be able to get to any assigned destination section from any starting section no matter what sections you are assigned to visit. If you are not able to get to every other section after starting at any section, print only the string "Oh, oh!" on the line. Otherwise print only the string "Gotta Catch Them All!" on the line.

Problem 6
Savage Pokémon (continued)

Sample Input

```
3
3 3
0 1
1 2
2 1
4 4
0 1
1 2
2 3
3 0
3 2
0 1
1 2
```

Output for the Sample Input

```
Oh, oh!
Gotta Catch Them All!
Oh, oh!
```

2016/2017 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 7
Spectral Composition

Scientists at the Interstellar Consortium of Planets and Constellations (ICPC) are studying the spectral composition of many celestial objects. The emission spectrum of a celestial object is the spectrum of frequencies of electromagnetic radiation emitted due to its atomic energy transitions. In other words it corresponds to the intensity for each color radiated by an object.

The ICPC stores the emission spectrum of a celestial object discretely by sampling it at different wavelengths. The emission spectrum is then represented with a list of integers where each position corresponds to the intensity of a specific wavelength. Contiguous positions in the list correspond to contiguous wavelengths in the spectrum.

Given an emission spectrum, the ICPC would like to know the intensity of the k-th wavelength, ordered by intensity, between two given wavelengths. However, in the lifetime of an object the intensity of two contiguous wavelengths could be swapped due to complex atomic reactions not yet fully understood. Your team is to write a program that will help the ICPC track wavelength swaps and determine the k-th most intense wavelength in a given range.

The input is in two sections. The first is the initial wavelength data which consists of a sequence of lines. Each line contains integers in the range 0–1,000,000 inclusive separated by whitespace. The first integer is w_0 and the last w_N. The wavelength section is terminated by an empty line. There will be at most 10 numbers on a line.

The remainder of the input consists of updates and queries, one per line. These are terminated by end of file.

An update has an "A" at the start of the line, a space, and an integer s, $0 \leq s \leq N-1$, and indicates that w_s and w_{s+1} have swapped places.

A query has a "Q" at the start of the line, a space, and three integers, i, j, and k separated by spaces. $0 \leq i \leq N, i \leq j \leq N, k \leq j - i$. The wavelengths of interest are w_i through w_j, and k indicates the k-th wavelength is the one wanted. For example, if $w_i \ldots w_j$ are $50, 27, 18, 0, 29, 18$, which ordered by intensity are $0, 18, 18, 27, 29, 50$, $k = 0$ would get 0 and $k = 5$ would get 50.

N will be at most 99,999 and the total number of updates and queries will be at most 1,000.

For each query print the w_{i+k} element after putting $w_i \ldots w_j$ in non-decreasing order. Print each number on one line with no unnecessary zeroes and no leading or trailing whitespace.

Sample Input

```
1 4 1 4 5 3 7

Q 0 6 2
Q 3 5 1
A 2
Q 3 5 1
A 5
Q 3 5 2
```

Problem 7
Spectral Composition (continued)

Output for the Sample Input

```
3
4
3
7
```

2016/2017 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 8
Hotel Rewards

You are planning to spend your holidays touring Europe, staying each night in a different city for N consecutive nights. You have already chosen the hotel you want to stay in for each city, so you know the price P_i of the room you'll be staying at during the i-th night of your holidays, for $i = 1, \ldots, N$.

You will book your accommodations through a website that has a very convenient rewards program, which works as follows. After staying for a night in a hotel you booked through this website you are awarded one point, and at any time you can exchange K of these points in your account for a free night in any hotel (which will however not give you another point).

For example, consider the case with $N = 6$ and $K = 2$ where the prices for the rooms are $P_1 = 10$, $P_2 = 3$, $P_3 = 12$, $P_4 = 15$, $P_5 = 12$ and $P_6 = 18$. After paying for the first four nights you would have four points in your account, which you could exchange to stay for free the remaining two nights, paying a total of $P_1 + P_2 + P_3 + P_4 = 40$ for your accommodations. However, if after the first three nights you use two of the three points you earned to stay the fourth night for free, then you can pay for the fifth night and use the final two points to get the sixth one for free. In this case, the total cost of your accommodations is $P_1 + P_2 + P_3 + P_5 = 37$, so this option is more cost-effective.

You want to write a program that determines what the minimum possible cost will be for your holiday accommodations. You should assume that all the hotels you want to stay at will have a room available for you, and that the order of the cities you are going to visit cannot be altered.

The first line of the input contains two integers N and K, separated by one or more spaces, representing the total number of nights your holidays will last, and the number of points you need in order to get a free night ($1 \leq N, K \leq 10^5$). The remaining N lines contain integers P_1, P_2, ..., P_N, representing the price of the rooms you will be staying at during your holidays ($1 \leq P_i \leq 10^4$ for $i = 1, 2, \ldots, N$). No input line will contain leading or trailing whitespace.

Your program is to print a line with a single value: the minimum cost of your accommodation for all of your holidays as an integer, without any leading zeroes or leading or trailing whitespace.

Sample Input

```
6 2
10
3
12
15
12
18
```

Output for the Sample Input

```
37
```

2016/2017 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 9
Ellipse

Your team is to write a program that will, given two ellipses, find the minimum distance between points on their respective perimeters. The ellipses will not intersect and one will not contain the other.

The equations for the general parametric form of an ellipse may be helpful:

$$x(\theta) = xc + a \cdot \cos(\theta) \cdot \cos(\phi) - b \cdot \sin(\theta) \cdot \sin(\phi)$$

$$y(\theta) = yc + a \cdot \cos(\theta) \cdot \sin(\phi) + b \cdot \sin(\theta) \cdot \cos(\phi)$$

where:

xc, yc: location of the center,
a: major axis radius,
b: minor axis radius,
ϕ: angle between x-axis and major axis, and
θ: the parameter $0 \leq \theta < 2\pi$. This is known as the the eccentric anomaly, as shown in Figure 1.

Figure 2 shows all of these values for the first ellipse in the sample input.

Input to your program is a series of lines terminated by end of file. Each line contains the definitions of the two ellipses as numeric values separated by whitespace:
$xc_1\ yc_1\ a_1\ b_1\ \phi_1\ xc_2\ yc_2\ a_2\ b_2\ \phi_2$

Values will fall within these limits:
$-10 \leq xc_1, yc_1, xc_2, yc_2 \leq 10$
$0.1 \leq a_1, b_1, a_2, b_2 \leq 10$
$b_1 \leq a_1$ and $b_2 \leq a_2$
$0 \leq \phi < 2\pi$

Input values will be either integers or floating point numbers without exponents.

For each pair of ellipses, your program is to print a line containing the minimum distance to 5 decimal places with no extra whitespace.

Sample Input

```
5 10 6 3 0.78539816339744828 -3 4 3 2 3.1415926535897931
5 10 6 3 0.78539816339744828 -3 14 3 2 3.1415926535897931
5 10 6 3 0.78539816339744828 10 18 20 2 0
1 2 10 .1 0  -1 -1 10 .1 0
1 2.000 8 4 0    -1 -5 10 2 0
```

Output for the Sample Input

```
1.36893
3.03640
1.26797
2.80100
1.02997
```

Problem 9
Ellipse (continued)

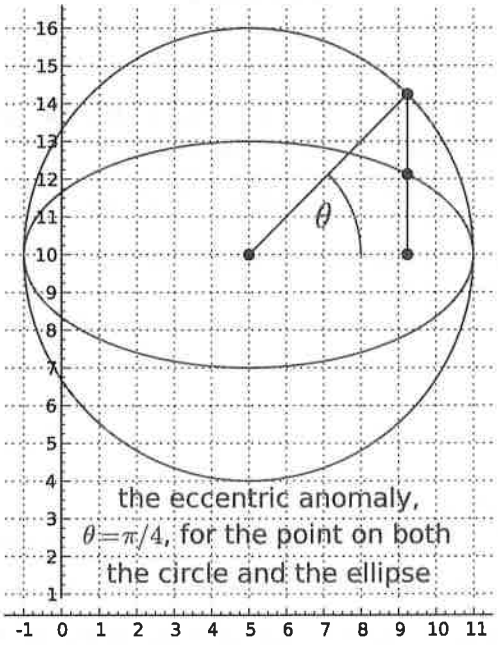

Figure 1. Construction of the Eccentric Anomaly.

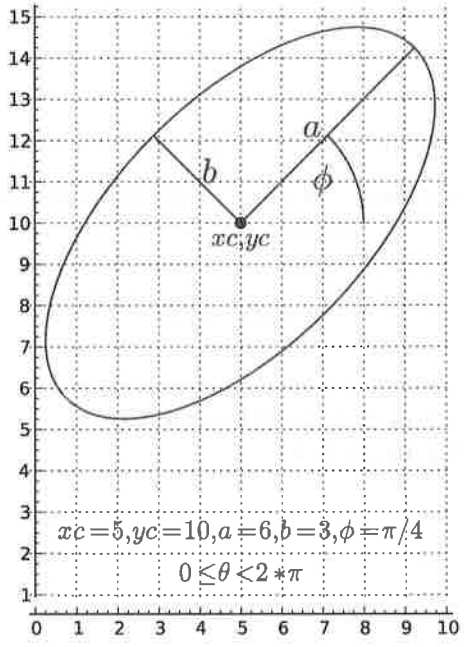

Figure 2. First ellipse in the Sample Input.

2016/2017 SOUTHERN CALIFORNIA REGIONAL ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 10
Lightweight Calendar App

Your team has been contracted to produce a very lightweight calendar mobile app. Given a sequence of iCalendar specifications interspersed with individual dates, your program must produce a single day's list of appointments based upon the calendar entries encountered in the input. An iCalendar specification is a series of *keyword:value* pairs, one pair per line as seen below:

`BEGIN:VCALENDAR`
`METHOD:`*methodname*
one or more event specifications
`END:VCALENDAR`

Events are single or multiple periods of time, called *occurrence(s)*, grouped together by a single, unique identifier (UID) as described below. *Methodname* can be either `PUBLISH`, `ADD`, or `CANCEL`. `PUBLISH` creates new calendar events (new UID), or completely supersedes the definition of existing events (matching a previously defined UID). `ADD` adds new occurrences to events with matching UIDs. `CANCEL` removes either occurrences from events, or entire events, for existing UIDs.

An *event specification* is itself a series of *keyword:value* pairs sandwiched between `BEGIN:VEVENT` and `END:VEVENT`:

`BEGIN:VEVENT`
event details
`END:VEVENT`

Each event is identified by a unique identifier (UID) specified within *event details*. *Event details* are also *keyword:value* pairs, one per line, and can be the following:

`UID:`*event id*
: unique event ID, occurs exactly once per `VEVENT`. 1–32 alphanumeric characters, possibly including hyphens

`SUMMARY:`*summary*
: text summarizing the event

`LOCATION:`*location*
: text describing the location of the event

`DTSTART[;` *optional timezone*`]:`*date-time*
: occurrence starting date and time, only in `PUBLISH` and `ADD` methods, exactly once per `VEVENT`

`DTEND[;` *optional timezone*`]:`*date-time*
: occurrence ending date and time, only in `PUBLISH` and `ADD` methods, mutually exclusive with DURATION

`DURATION:`*length of time*
: event duration, only in `PUBLISH` and `ADD` methods, mutually exclusive with DTEND

`SEQUENCE:`*sequence number*
: if omitted, defaults to the highest specified sequence, starting at 0

`RDATE:`*list of dates*
: an optional comma-separated list of one or more dates that the event will occur on.

`RECURRENCE-ID[; THISANDFUTURE]:`*individual occurrence start date-time*
: for `CANCEL` method, the starting date-time corresponding to an occurrence. If `THISANDFUTURE` is specified, remove the occurrence identified by date-time, as well as all occurrences after date-time. The absence of `RECURRENCE-ID` in a `CANCEL` method indicates removal of the entire event.

Problem 10
Lightweight Calendar App (continued)

Your program should process the above keywords *only* when they occur between `BEGIN:VEVENT` and `END:VEVENT`. Ignore any timezone specifications (all times will be for a single timezone). All dates will be in a Gregorian Calendar system.

The fields in the previous descriptions are defined as follows:

summary and *location*
: printable (non-control) characters except double-quote, comma, semicolon, and backslash. Maximum of 32 characters. *Summary* will not include lower-case letters.

date-time
: string in the form *YYYYMMDDThhmmss*, where *YYYY* is a four digit year, *MM* is a month in the range 01–12, *DD* is a day in the range 01–31, *hh* is hours 00–23, *mm* is minutes 00–59, and *ss* is seconds 00–59. There will be no leap seconds in the input. All dates will be legal dates. You can expect leap years to be present.

date
: string in the form *YYYYMMDD*, where *YYYY*, *MM*, and *DD* are described as for *date-time*.

length of time
: string of the form *Thhmmss*, where *hh* is hours 00–23, *mm* is minutes 00-59, and *ss* is seconds 00-59, specifying the length of an occurrence of the event.

sequence number
: an integer, used in combination with a UID. If a `VEVENT` specification matches a previously encountered UID, and the sequence number is greater than the highest value previously encountered, then the new specification takes precedence. If the *sequence number* is less than or equal to the previously encountered `VEVENT` sequence number, then the update is out of sequence and should be ignored.

Occurrences within a single event (same UID) will never overlap. Different events (different UIDs) can have overlapping occurrences. There will be no zero-length occurrences. For occurrences that end at midnight, the occurrence is reportable only for the preceding day(s). For example, an event start date and time of November 12, 2016 at 1900 with a 5 hour duration should only be reported for November 12th.

Input to your program is a list of `VCALENDAR` entries, interspersed with report dates. There are many lines with *keyword:value* pairs not specified here. Your program must ignore such lines. A report date is a single line of the form *YYYYMMDD*, just like the format of *RDATE* dates. *YYYY* will be in the range 2000–2800, and leap years can occur. A year is a leap year if it is a multiple of 400, or if it is a multiple of 4 and not a multiple of 100.

Your program is to print one or more lines per reporting date specified. On the first line, print the date as *MM-DD-YYYY* followed by end-of-line. *MM* is the two-digit month, *DD* is the two-digit day, and *YYYY* is the four-digit year. Months and days should be printed with leading zeroes if needed.

Your program should then print a line for each occurrence that spans any time within the reporting date, with start and stop times for only the times that occur within the reporting date. The line starts with two spaces, followed by occurrence information of the form "*HH:MM:SS-hh:mm:ss summary (location)*" where *HH:MM:SS* is the starting time and *hh:mm:ss* is the ending time.*HH* and *hh* are the two-digit hours (00–23), *MM* and *mm* are the two-digit minutes, and *SS* and *ss* are the two-digit seconds. Hour, minute, and second values should be printed with a leading zero if needed. For occurrences that end at or after 00:00:00 on the subsequent day, print a stop time of 24:00:00.

The *summary* is the the text from the `SUMMARY` keyword associated with the event. *location* is the text from the `LOCATION` keyword. If no summary or location text exists, substitute no characters into the format, but leave the format the same. For example, no summary would produce two spaces between the stop time and the open parenthesis, whereas no location would produce "()".

Problem 10
Lightweight Calendar App (still continued)

Print the occurrences in ascending order by start time. For occurrences with the same start time, print the shorter one first. For occurrences with the same start and end times, print the occurrence with the lexicographically sorted `SUMMARY` first.

Sample Input

```
BEGIN:VCALENDAR
METHOD:PUBLISH
BEGIN:VEVENT
SUMMARY:PRACTICE SESSIONS
DTSTART;TZID=America/Los_Angeles:20160913T150000
DTEND;TZID=America/Los_Angeles:20160913T160000
DTSTAMP:20160914T031535Z
UID:09664996-2cd8-4ee1-a8e3-7ec2bb1c23b8
SEQUENCE:1891
LOCATION:Library
RDATE:20161101,20161108,20160920,20160927,20161004,20161011,20161018,20161025
END:VEVENT
BEGIN:VEVENT
SUMMARY:CONTEST
DTSTART;TZID=America/Los_Angeles:20161112T090000
DTEND;TZID=America/Los_Angeles:20161112T210000
DTSTAMP:20160914T031535Z
UID:96e4f592-8a4e-42de-8826-d377b52d9f47
SEQUENCE:1891
LOCATION:RCC
END:VEVENT
END:VCALENDAR
BEGIN:VCALENDAR
METHOD:ADD
BEGIN:VEVENT
UID:09664996-2cd8-4ee1-a8e3-7ec2bb1c23b8
DTSTART;TZID=America/Los_Angeles:20160921T210000
DURATION:T050000
SEQUENCE:1892
END:VEVENT
END:VCALENDAR
20160920
20160922
BEGIN:VCALENDAR
METHOD:CANCEL
BEGIN:VEVENT
UID:09664996-2cd8-4ee1-a8e3-7ec2bb1c23b8
SEQUENCE:1893
RECURRENCE-ID:20160920T150000
STATUS:CANCELLED
END:VEVENT
END:VCALENDAR
20160920
```

Problem 10
Lightweight Calendar App (still continued)

Output for the Sample Input

```
09-20-2016
  15:00:00-16:00:00 PRACTICE SESSIONS (Library)
09-22-2016
  00:00:00-02:00:00 PRACTICE SESSIONS (Library)
09-20-2016
```