

# 2000 ACM Southern California PC

0. [MET Conversion](#) (warm-up)
1. [SPC](#)
2. [Map Corners](#)
3. [Swamp Router](#)
4. [People First](#)
5. [Vote Count](#)
6. [Team Packing](#)

**2000/2001 ACM SOUTHERN CALIFORNIA REGIONAL  
SCHOLASTIC PROGRAMMING CONTEST**

**Problem 0  
MET Time Conversion**

During a spaceflight mission, it is often convenient to schedule activities in terms of elapsed time since the start of the mission, which is usually the time of launch. Times in this format are said to be in Mission Elapsed Time (MET) and are expressed in terms of days, hours, minutes, seconds, and (sometimes) fractional seconds. Given as inputs a reference (launch) time and a list of times in MET format, write a program that outputs the corresponding calendar date and time for each MET time.

*Input*

The reference time will be expressed as a date and time of day in format MM/DD/YYYY hh:mm:ss, where

MM	month, expressed as a 2-digit number in the range [01, 12]
DD	day of month, expressed as a 2-digit number in the range [01, 31]
YYYY	4-digit year in the range [1950, 2199]
hh	hours, expressed as a 2-digit number in the range [00, 23]
mm	minutes, expressed as a 2-digit number in the range [00, 59]
ss	seconds, expressed as a 2-digit number in the range [00, 59]

and the time itself is in Greenwich Mean Time (GMT). You can assume that the reference date will be legal (e.g., April 31 will not be an input). The date and time of day will be separated by one blank and will appear on a single line with no leading or trailing whitespace.

The remaining lines of input will contain the input MET times, each of which will be on a single line. The input MET times will be expressed as [-]DD/hh:mm:ss, where an optional negative sign [-] denotes a time prior to the reference time, DD is a 2-digit number in the range [00, 99], and hh, mm, and ss are defined as shown above. For simplicity, you can also assume that missions do not cross year boundaries; i.e., the “real” calendar date/time corresponding to each input MET time will always fall within the same year as the reference time.

Missions are allowed to occur in leap years. For the specified range of years, a year is a leap year if it meets one of the following two conditions: (1) it is an integer multiple of 4 and not an integer multiple of 100; or (2) it is an integer multiple of 400. For this problem, you can ignore the existence of leap seconds.

*Output*

For each input MET time, output the corresponding calendar date/time in GMT on a single line in the same format as that used for the reference time. Do not add leading or trailing whitespace to the output.

*Sample Input*

```
02/11/2000 17:43:40
09/12:10:52
-00/00:00:01
```

*Sample Output*

```
02/21/2000 05:54:32
02/11/2000 17:43:39
```

**2000/2001 ACM SOUTHERN CALIFORNIA REGIONAL  
SCHOLASTIC PROGRAMMING CONTEST**

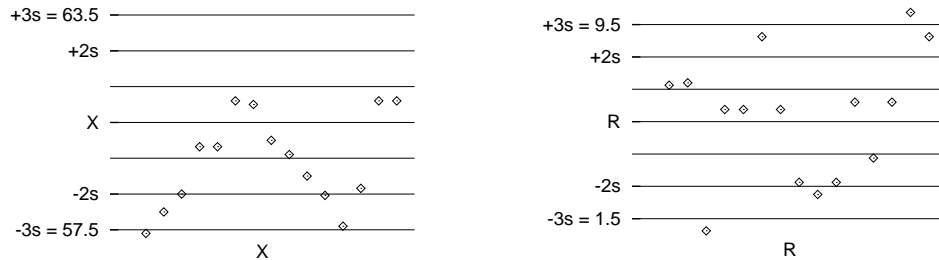
**Problem 1  
Statistical Process Control**

Swamp County Electronics is a leading supplier of military-grade widgets, and has been using Statistical Process Control for years to help monitor and control its widget manufacturing process. Statistical Process Control, or SPC, is a discipline that uses sampled and measured products, statistics, and historical sample data to determine if a manufacturing process is either in control or out of control.

Swamp County Electronics (SCE) uses *control charts* to detect when its product (hence its process) goes out of control: at the end of each hour of production, five units are removed from the assembly line for measurement. Statistics calls the taking of one or more measurements for a single statistical analysis a *sample*, usually denoted  $X$ . Testing the widgets is very complicated; only two units can be tested per hour. The testing apparatus is outrageously expensive and SCE can afford only one such apparatus. As such, at the end of the eight hour workday, the Quality Assurance Team works into the wee hours of the morning, measuring all the units in each sample. As soon as each sample is measured, the sample mean,  $\bar{X}$ , and the sample range,  $R$ , are calculated:

$$\bar{X} = \frac{1}{5} \sum_{i=1}^5 x_i \qquad R = \max(X) - \min(X)$$

where  $x_i$  represents each individual measurement.  $\bar{X}$  and  $R$  are then plotted consecutively on an X&R control chart as in Figure 1. The centerlines,  $\bar{\bar{X}}$  and  $\bar{\bar{R}}$ , are the process targets determined previously by more in-depth analyses. The control limits are set at  $\pm 3s$ , where  $s$  is the expected standard deviation for the process, again determined previously.

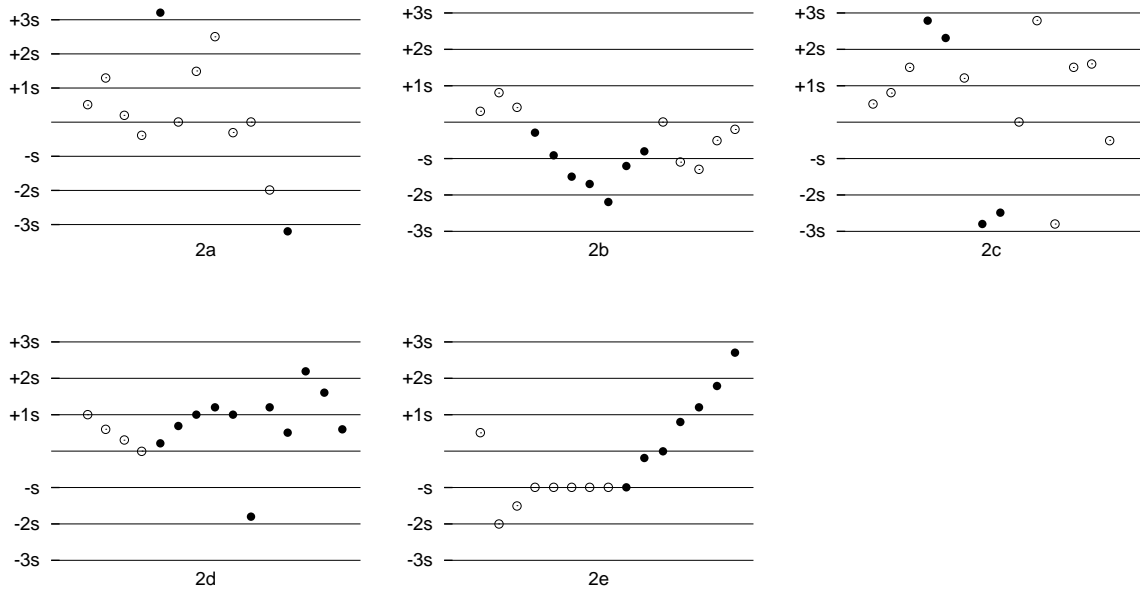


**Figure 1.** X&R chart from the sample input.

When one of more of the out-of-control patterns in Figures 2a through 2e are detected on either of the X or R control charts, production is stopped until the cause of the irregularity is fixed, and *all* of the units manufactured since the *onset* of the irregularity are discarded. The problem is that the irregular patterns can take up to ten hours to detect, and may not be detected until two days after the onset of the problem.

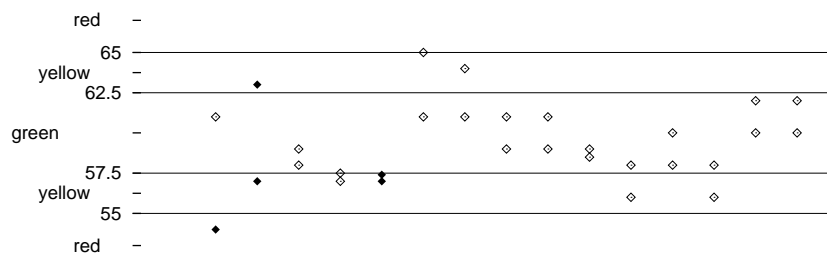
Before investing in two more test apparatuses to detect irregularities in “real time,” SCE wishes to investigate the use of *precontrol charts*. Precontrol charts require only two measurements per sample, and these measurements are plotted with respect to the specified product tolerances (upper and lower limits) and precontrol, or PC, lines. PC lines are set at  $\frac{1}{4}$  and  $\frac{3}{4}$  of the total specified tolerance range. See Figure 3. The precontrol chart is divided into red, yellow, and green zones, with the red zone outside the tolerance limits, yellow between the PC line and the nearest tolerance limit, and green between the PC lines. Green is defined by the closed interval  $[0.25, 0.75]$  of the tolerance range, yellow by the half-opened intervals  $[0, 0.25)$  and  $(0.75, 1]$  of the tolerance range. If either measurement falls in a red zone, or both measurements fall into yellow zones, the process is probably out of control. Compared to X&R charts, the precontrol charts have a higher probability of signaling a problem incorrectly (a false positive), or failing to detect a problem even if it exists (false negative).

**Problem 1**  
**Statistical Process Control (continued)**



**Figure 2.** Out of control sequences for  $\bar{X}$  or  $R$ .

Figure 2a: a single point outside of  $\pm 3s$ . Figure 2b: seven successive points on one side of the centerline; *a point on the centerline cancels the sequence*. Figure 2c: two successive points in  $[-3s, -2s)$  or  $(+2s, +3s]$ . Figure 2d: exactly 10 out of 11 successive points on one side of the centerline; *a point on the centerline cancels the sequence*. Figure 2e: a trend of seven successive points upward or downward (a point followed by six monotonically increasing or decreasing points).



**Figure 3.** Precontrol chart from the sample input.

As a precursor to a cost-benefit analysis, SCE has contracted your team to simulate the precontrol chart's effectiveness compared to the X&R chart's trusted behavior. You will use the historical sample data from the control charts. From this data you will compute sample means and ranges for control chart analysis. Then, using only the first two measurements of each sample, you will apply the precontrol chart criteria (red, yellow, green zones), indicating when a false positive or false negative occurs with respect to the out-of-control patterns depicted in Figures 2a through 2e. Bear in mind that the control chart method is a series-based indicator—when an irregularity is detected at the end of a sequence, the irregularity actually started at the beginning of the sequence. The precontrol chart indicates “instantaneous” behavior. *A precontrol false positive occurs when there are no irregular control chart sequences overlaying its sample, and occurs at the instantaneous sample.* A precontrol false negative really doesn't exist at a particular sample; rather it is the lack of any precontrol problem indication over the duration of a control chart out-of-control sequence. *A false negative is reported at the end of a control chart irregular sequence.*

**Problem 1**  
**Statistical Process Control (still continued)**

Input to your program consists of two lines of quality specifications: the first line contains the process mean,  $\bar{X}$ , and upper and lower control limits. The upper and lower control limits are always centered about  $\bar{X}$ . The second line contains the range average,  $\bar{R}$ , and its upper and lower control limits, centered about  $\bar{R}$ . The third line contains the upper and lower product tolerances. Each remaining line until end-of-file holds the data for a five-measurement sample. All measurements fall within the range [-1000,1000]. Only the values measured may be viewed by your team; what the actual data signifies is highly classified due to the military nature of the production contract. All values within lines are separated by whitespace. The samples are numbered implicitly: line 4 is sample 1, line 5 is sample 2, etc.

Output is a list of false positive or false negative indications along with the sample number where they occurred, one indication per line. In either case, output the sample number without any leading spaces. For a false positive, follow the sample number with the string "false positive". For a false negative, follow the sample number with the string "problem missed". The output must appear in ascending order of sample number.

*Sample Input*

```
60.5 63.5 57.5
 5.5  9.5  1.5
65 55
54  61  59.5 56  56.5
63  57  57  57.1 55.9
58  59  58  59  58.5
57  57.5 63  62.5 59.1
57  57.4 63  62.5 59.2
65  61  56  61.5 62
64  61  58  60  62
59  61  58.5 60  61.5
61  59  58.5 60.5 59
58.5 59  61  58.5 58
56  58  58  62.3 58
58  60  57  56  57
58  56  59  62.3 58
60  62  56  61.5 66
60  62  56.5 61.5 65.5
```

*Sample Output*

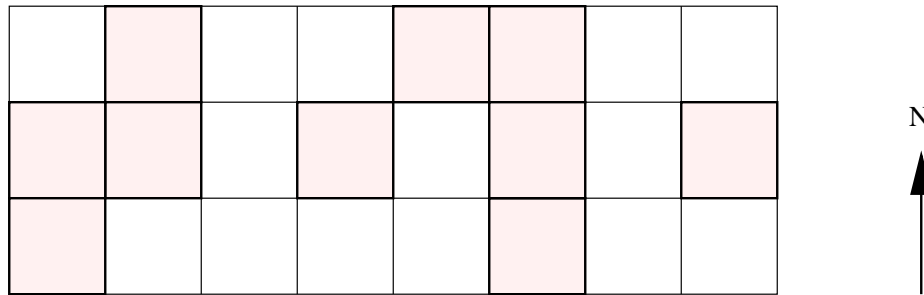
```
2 false positive
3 problem missed
5 false positive
12 problem missed
14 problem missed
```

**2000/2001 ACM SOUTHERN CALIFORNIA REGIONAL  
SCHOLASTIC PROGRAMMING CONTEST**

**Problem 2  
Map Corners**

Satellite imagery that was collected worldwide is being reformatted into a latitude/longitude coordinate system in order to produce a map. The data is being divided into square cells for layout onto a grid. The map grid extends completely around the world in the longitudinal (west-east) direction but does not extend to either the North or the South Poles in the latitudinal (north-south) direction. Within the area covered by the grid, there are cells for which no data was collected. These cells are to be left blank on the map.

All cells are of equal size and are oriented so that their corners coincide with the intersections of the gridlines. Figure 1 shows a sample map grid with 8 cells in the longitudinal direction and 3 cells in the latitudinal direction. The map wraps around at the left and right edges. The areas containing data are represented by shaded cells. In the following discussion, the shaded cells are referred to as map cells.



**Figure 1.** Sample map grid.

During the map-production process, it will be necessary to identify all points on the grid that are shared by more than one map cell. You are to write a program that reports all such points on the grid and the number of map cells sharing each point.

*Input*

The first line of input will contain the dimensions  $L$  and  $M$  of the map, represented as the numbers of cells in the longitudinal ( $x$ ) and latitudinal ( $y$ ) directions, respectively. Both numbers will be integers separated by whitespace. You can assume that  $L$  is an integer in the range  $[3, 720]$  and  $M$  an integer whose value is constrained by  $L$  and the requirements on the map.

The second line of input will contain the total number of map cells,  $P$ . You can assume that  $P$  is a positive integer that does not exceed the maximum allowed by the map dimensions.

The final  $P$  lines of input will be a list of the map cells. Each cell will be identified by the (integer) grid coordinates of its southwest (lower left) corner on the map, where  $(0, 0)$  is defined to be the coordinates of the southwest corner of the grid. Each cell is one unit wide in the  $x$ -dimension and one unit high in the  $y$ -dimension. One cell will be identified per line of input. On each line, the  $x$ -coordinate will be specified first, followed by whitespace, followed by the  $y$ -coordinate. The cells will be ordered randomly. You can assume that the map cells will be within the bounds of the defined map, and that the  $x$ -coordinates are constrained to the range  $[0, L - 1]$ .

**Problem 2**  
**Map Corners (continued)**

*Output*

Your program should list all the unique grid points that are shared by at least two map cells. For each shared grid point, output on a single line the point's  $x$ -coordinate,  $y$ -coordinate, and the number of map cells sharing that point, all separated by one blank and with no leading or trailing whitespace on the line. Output the grid points (if any) along the southern border of the map first, in order from west to east, starting at  $x$ -coordinate 0; continue in the same manner along successive lines of latitude, ending with the northern border of the map. If no shared grid points can be found in the entire map, your program should print the string

NO SHARED CORNERS

and exit. Note that since the map wraps around the world at its eastern border, points along the map's eastern border are the same as those along its western border and should not be reported twice.

*Sample Input*  
Refer to Figure 1.

```
8 3
10
0 0
0 1
1 1
1 2
3 1
4 2
5 2
5 1
5 0
7 1
```

*Sample Output*

```
0 1 3
1 1 3
5 1 2
6 1 2
0 2 2
1 2 3
2 2 2
4 2 2
5 2 3
6 2 2
5 3 2
```

**2000/2001 ACM SOUTHERN CALIFORNIA REGIONAL  
SCHOLASTIC PROGRAMMING CONTEST**

**Problem 3  
Swamp Router**

Your competitors in the low end routing business are adding firewall type features and your sales department is frantic. Sales wants to add access lists ASAP. The marketeers also require that 'wire speed' routing not be compromised, but the routing engine is underpowered. Your job is to write a simulator to help answer the performance questions.

Engineering has used a sniffer to record network traffic and to extract the relevant information from the packet headers, generating a stream of simulated traffic. Here are some samples of the simulated traffic packets:

```
216.35.137.204 0 204.69.7.5 0 1
204.69.3.100 49152 204.69.7.109 53 17
24.218.179.217 27015 204.69.0.4 2326 17
148.246.129.175 6699 204.69.2.85 1138 6
```

The packet fields are:

<i>Field Number</i>	<i>Contents</i>	<i>Format</i>
1	source IP address	dotted decimal address
2	source port	decimal number
3	destination IP address	dotted decimal address
4	destination port	decimal number
5	protocol	decimal number

The fields are separated by one or more blanks and the first field may be preceded by one or more blanks. A dotted decimal address is a 32-bit value, high order byte first, with the decimal representation of each 8-bit byte [0..255] separated by a '.'. For example, 1.2.255.15 is 0x0102FF0F. Ports are in the range [0..65535]. Note that ports are significant for the TCP (6) and UDP (17) protocols only. The port fields will be 0 for other protocols. The protocol is in the range [0..255].

You will be given sample access lists. They will be syntactically correct with fields that are separated by one or more blanks. Some example entries are:

```
* 1.2.3.4 255.255.0.15 3 5.6.7.8 0.0.255.255 9 Permit
6 0.0.0.0 0.0.0.0 * 204.69.4.254 255.255.255.255 0-1023 deny
```

The access list fields are:

<i>Field Number</i>	<i>Contents</i>	<i>Format</i>
1	protocol	decimal number or '*'
2	source IP address	dotted decimal address
3	source mask	dotted decimal mask
4	source port	decimal number or '*' or a range
5	destination IP address	dotted decimal address
6	destination mask	dotted decimal mask
7	destination port	decimal number or '*' or a range
8	action	'permit' or 'deny' (case is not significant)

### Problem 3

#### Swamp Router (continued)

A range is two decimal numbers separated by '-' with no whitespace between the numbers and the '-'. For example, 5-52 will match  $5 \leq \text{port number} \leq 52$ . In the access list, the protocol and the ports can be specified as '\*', which means match any protocol or port.

A dotted decimal mask is the same format as an address. It indicates which bits are significant when comparing addresses. For example, the addresses 123.234.248.14 and 123.234.254.126 match when compared using the mask 255.255.240.15 (0xFFFF00F). The 1 bits in the mask show which bits in the addresses to compare. Note that a mask of 0.0.0.0 will cause any address pair to match.

Each simulated traffic packet is compared with the access list entries, in the same order those entries were entered, until the packet matches an entry. For a match to occur, the protocol, source and destination addresses (using the respective masks), and ports must *all* match. In the real router, the action directs the router to accept or reject the packet, but the simulator just keeps track of how many access list entries are compared, and the number of times each action is taken. For example, if a packet matches the 3rd, 5th, and 8th entries in a 30-entry access list, with the action of the 3rd entry being 'permit', then processing that packet will contribute 3 comparisons and add 1 to the number of 'permit' actions taken. If the action were 'deny', 3 would also be added to the comparison count, but 1 would be added to the 'deny' count. In either case the 5th and 8th entries are not examined. The first entry that matches is the last examined for a given packet.

#### *Input*

Input is a series of test cases. Each test case consists of the access list entries terminated by a blank line. The simulated packet traffic data follows. Each line will be at most 100 characters. The test case is terminated by a blank line or end-of-file. Each test case will have at most 100 access list entries.

#### *Output*

For each test case print three numbers: the total count of access list entries compared, the number of simulated packets resulting in the 'permit' action, and the number of simulated packets resulting in the 'deny' action. Put no leading spaces before the first number and exactly one space before the second and third numbers.

**Problem 3**  
**Swamp Router (still continued)**

*Sample Input*

```
* 204.69.0.0 255.255.248.0 * 0.0.0.0 0.0.0.0 * permit
* 0.0.0.0 0.0.0.0 * 0.0.0.0 0.0.0.0 * DENY

204.69.6.230 3072 24.115.85.245 23 6
192.168.215.17 53 204.248.52.7 1243 17
204.69.12.21 2118 152.163.241.11 21 6

* 0.0.0.0 0.0.0.0 * 0.0.0.0 0.0.0.0 * deny

192.102.199.19 27961 63.20.61.96 65535 17

6 198.138.176.100 255.255.255.255 * 204.69.4.100 255.255.255.255 111 permit
6 0.0.0.0 0.0.0.0 * 204.69.0.0 255.255.240.0 111 deny
6 0.0.0.0 0.0.0.0 * 204.69.0.0 255.255.248.0 80 permit
* 0.0.0.0 0.0.0.0 * 204.69.0.255 255.255.248.255 * deny
* 204.69.0.0 255.255.248.0 * 0.0.0.0 0.0.0.0 * deny
* 127.0.0.0 255.0.0.0 * 0.0.0.0 0.0.0.0 * deny
* 0.0.0.0 0.0.0.0 * 204.69.7.240 255.255.255.240 9990-9999 deny
* 0.0.0.0 0.0.0.0 * 0.0.0.0 0.0.0.0 * permit

204.69.4.87 80 204.69.9.12 6776 6
134.79.112.65 2186 204.69.4.218 21 6
216.112.217.140 2212 204.69.5.255 22 6
198.138.176.100 1053 204.69.4.100 111 6
198.138.176.100 1054 204.69.5.100 111 6
```

*Sample Output*

```
5 1 2
1 0 1
20 2 3
```

**2000/2001 ACM SOUTHERN CALIFORNIA REGIONAL  
SCHOLASTIC PROGRAMMING CONTEST**

**Problem 4  
PeopleFirst**

The human rights organization, PeopleFirst, has observers around the world checking up on human rights abuses in various countries. Since some countries take a dim view of this sort of activity, PeopleFirst needs a way for their observers to report abuses without the countries they are residing in realizing that a report is being made. The idea is this: Their observers will create websites where they will display photographs of their sightseeing trips, collections, or other hobbies, but the pictures will be subtly altered to hide the reports in them.

There is software available to do this, but PeopleFirst doesn't want to run the risk that some government will try one of these programs on their observer's pictures and find a hidden message. To avoid that, one of their members has written a custom program for hiding messages in pictures. Unfortunately, she had to leave on her own stint as an observer before she could write the program that extracts the messages from the pictures. That's where you come in. You've been asked to write the missing program.

The observer's photographs are posted to their web sites in GIF format. Prior to being given to your program, the pictures are run through a filter that converts them to ppm format, so your program has a simpler picture representation to work with. The ppm format begins with the two characters "P6" followed by whitespace, most often a single linefeed character. This is followed by the picture width and height formatted as two decimal numbers in ASCII, separated by whitespace. The height is followed by more whitespace, again most often a single linefeed. Following this is the maximum color component value: a decimal number in ASCII. This is the value of the brightest shade for any of the three primary colors in a pixel. You should ignore this value and treat it as if it were 255. This is immediately followed by a single whitespace character, usually a linefeed. Immediately after the whitespace character are  $3 \times width \times height$  bytes representing a left to right, top to bottom scan of the pixels in the picture, three bytes per pixel, representing the shades of red, green and blue for the pixel, in that order. Treat each byte as an unsigned binary number representing the shade.

The message is hidden in the picture by altering the original pixel values slightly. Each pixel in the picture encodes one bit of the message, which is extracted by taking the exclusive-or of the low order bits of the shades of red, green and blue that make up the pixel. Message bits are assigned to pixels from left to right in rows, starting with the top left pixel in the picture, and ending with the bottom right pixel. The first thing encoded in the picture is the length of the message. It is encoded as 32 bits of binary, starting with the low order bit. The length is immediately followed by the bytes of the message, each byte encoded with the low order bit first. The message is immediately followed by a 32-bit checksum, with the low order bit appearing first.

The checksum is calculated from all of the bytes of the message length and the message. In calculating the checksum, the bytes of the message length are ordered with the low order byte first. Initialize the checksum to hexadecimal ffffffff. For each byte that goes into the checksum, first do a single bit left circular shift of the 32 bits of the checksum, then add the byte as an unsigned value, ignoring any overflow.

Output consists of the extracted message, or an error message if no hidden message was found in the input picture. A bad message length or invalid checksum indicate that the input picture has no hidden message.

If the extracted message length is zero, output the following error message and exit before producing any other output:

**No message found: message length is zero**

## Problem 4 PeopleFirst (continued)

If the extracted message length has a value so large that there wouldn't be enough pixels in the input picture to encode all of the bits of the message length, message and checksum, output the following error message and exit before producing any other output:

```
No message found: bad length:  msz bits >  psz pixels
```

In the actual message, replace *msz* with the number of bits needed to encode the message length, message and checksum; replace *psz* with the number of pixels in the picture. Output both numbers in decimal.

If the checksum you calculate does not equal the extracted checksum, output the following error message and exit before producing any other output:

```
No message found: bad checksum: recorded =  rcs, calculated =  ccs
```

In the actual message, replace *rcs* with the extracted checksum, and replace *ccs* with the calculated checksum. Output both checksums with the prefix "0x", followed by 8 hexadecimal digits, using lower case for the alphabetic hexadecimal digits. The string "0x13579bdf" would be one example of the required format.

The judge's data will have no improperly formatted input or require impossibly large memory allocations. Consequently, the three defined errors will be the only ones that can appear legally in your output.

Note that the ppm formatted picture is likely to contain many unprintable characters, and there is no requirement that the message consists solely of printable characters either. The sample input printed on this page has `␣` to indicate spaces and bullets (•) that represent `0x0a`. Each line is broken after the occurrence of `0x0a` for readability. To view your input and output, the `od` command should prove useful. To view the pictures as pictures, you can use `xloadimage`. A more interesting example is available via the `getdata` command: the picture, `jellies.ppm`, hides another picture, the GIF file, `redrose.gif`.

### Sample Input

```
P6•
8␣12•
255•
!~␣~p␣}‘␣~P␣~@␣~0␣~␣␣~•
0~~0~p0~‘0~P0~@0~/~!0~•
)}~@~p@~‘@~P@~@@~0@~␣@~•
P~~P~pP~‘P~PP~@P~OP~␣P~•
‘~‘~p‘‘‘‘Q‘‘@‘‘0‘‘!‘‘~•
o}~o~qp~‘p~0p~@p~/p~!p~•
~q~pp~p‘~pP~p@~p1~p␣~p•
~‘~‘ap~‘‘‘~_P~‘@‘‘0‘‘␣~‘•
~P~~Pp~P~‘~PP~P)}Q1~0~_~Q•
~A~@p~@a~@P~@@~@0~@␣~@•
~0~Op~0‘~OP~0@~00~0␣~0•
~␣~␣p~␣‘~␣P~␣@~␣0~␣␣~"•
```

*Note the space characters `␣` and `0x0a` •*

### Sample Output

Hi!

**2000/2001 ACM SOUTHERN CALIFORNIA REGIONAL  
SCHOLASTIC PROGRAMMING CONTEST**

**Problem 5  
Vote Count**

Swamp County has gotten new hardware for handling and reading ballots, so they need to replace their vote counting software. Frugal as always, the county supervisors have found volunteer labor, you, to develop the software, so they can use the savings as justification for voting themselves hefty raises as a reward for saving the county so much money.

Here's what you have to do to fulfill your civic duty: Write a program that accepts a description of the choices before the voters in the election and how those choices are represented on the ballot, reads the raw ballots, and prints the results.

To avoid wordiness, we will use the word "question" to represent the office, proposition, or other question put before the voters, and the word "choice" to represent the candidates, "yes" or "no," or the other choices voters can make on a question. The ballots used in Swamp County have pre-recorded precinct numbers, and contain 384 positions (numbered from 1 to 384) that can be used to record voters' choices. Only a small number of these positions will be used in any election to represent choices on ballot questions.

Input is in three sections, each of which consists of several lines. Every line will contain one or more data items, and is terminated by a newline character. The format of the data items is described later.

The first section is a description of the ballot questions. This section begins with a line containing a single number which gives the total number of ballot questions. It is followed by the descriptions of the questions themselves, each in its own subsection. Questions are numbered, starting with 1, in the order that they appear in this section.

The first line of the subsection for a ballot question contains its name. The second line contains two numbers, the first of which is the number of choices for this question. The second number is the number of choices that can be voted for on the question, since some questions, like positions on a party's central committee, allow you to vote for more than one candidate. The remaining lines are the names of the choices, one per line. The choices are numbered from 1, in the order they appear in the question.

The second section is a description of the ballots, which has a separate subsection for each precinct, since which questions are voted on and the order of the choices may vary by precinct. The first line in this section contains the number of precincts in the county. It is followed by the subsections describing the ballots in each precinct.

The first line of a precinct ballot description subsection contains two numbers: the precinct number, and the number of ballot positions in use on ballots in that precinct. The following lines associate one of those ballot positions with the vote it represents. Each line contains a ballot position number and the question and choice numbers associated with that ballot position. Thus a line containing "31 2 5" indicates that a vote in position 31 on that precinct's ballot is a vote for choice 5 of question 2, as given in the description of the ballot questions in the first section of input.

The last section is the raw ballot information, one line per ballot, until end-of-file. A ballot line starts with the precinct number for the ballot, right justified in a 4-character field. This field is followed (after a single blank) by 48 pairs of hexadecimal digits encoding the 384-bit bit string that represents how the ballot was voted. In this representation, the bits in the bit string are numbered from 1 to 384, and each position on the ballot is represented by the identically numbered bit. That bit is 1 if the corresponding ballot position is marked, and 0 if it is not marked. In the string of 96 hexadecimal digits that encodes this bit string, bit number 1 is the high order bit of the first hex digit, and the numbering proceeds from high order to low order bit in each hexadecimal digit, until bit number 384, which is the low order bit of the last hex digit. Count the ballots, starting at 1 as you read them, in case you need to report an error on that ballot.

## Problem 5 Vote Count (continued)

Numbers are represented by strings of decimal digits in ASCII. Bit strings are represented by strings of pairs of hexadecimal digits, as already described. Multiple numbers and bit strings can appear on the same line, separated by single blanks. Character strings are represented by a single line of printable characters, including blanks, but excluding the newline character that terminates the line.

Ballots must be checked for validity before they can be counted. If the precinct number on a ballot is not known (did not appear in the ballot description section of the input), print the ballot number and the string “bad precinct” as an error message. If a ballot has any marks in positions that do not correspond to any ballot choices for ballots in that precinct, print the ballot number and the string “stray mark” as an error message. Either of these two errors will void the entire ballot. Do not count any votes from such a ballot.

If a ballot has votes for more than the allowed number of choices on a question, only the vote on that question is voided. This error does not void the entire ballot. If this happens, print the ballot number, the question number, and the string, “too many votes” as an error message. Do not count the vote for any choice on that question (or any other voided question) on the ballot, but do count the votes for any questions that have not been voided on that ballot.

Each error message should be on a single line and should contain just the numbers and strings mentioned. There should be just a single blank character between each number and string. Do your validity checks for each ballot in the order given. If the precinct is bad, don’t bother checking anything else. If there are stray marks, don’t bother reporting that some questions may have too many votes. For any ballot, report too many votes on more than one question in the order of their question numbers. If there are any error messages, follow them by three empty lines to separate them from the result output.

The first line of result output should start with 32 blanks, followed by the string “Election Results”, providing a title for the election results. The title line is followed by an empty line and then the results for each question in the order they were presented in the first section of the input.

The result for each question is output as a line containing 8 blanks and the name of the question, followed by an empty line, followed by one line for each choice in that question, in the order they appeared in the first section of the input. The line for each choice begins with the choice name, left justified in a field of 64 characters, followed by a single space followed by the number of votes for that choice, right justified in a field of 10 characters, followed by a single space, followed by the percentage of the vote for that choice on that question, rounded to the nearest integer, right justified in a field of 3 characters, and followed by a percent sign. Note that the percentage is the percentage of ballots cast on that question that voted for that choice. That means that, if you can vote for 2 choices on that question, the percentages, subject to rounding error, can add up to 200%. Always round .5 up.

The output of the results for the ballot questions should be separated from each other by single empty lines, and there should be no empty line after the results for the last question.

No input line in the ballot description sections is longer than 64 characters, while every raw ballot is represented by a line that is exactly 101 characters long. The line lengths do not include the line terminator.

**Problem 5**  
**Vote Count (still continued)**

*Sample Input*

3  
President  
5 1  
Bugs Bunny  
Daffy Duck  
Goofy  
Kenshiro  
Mickey Mouse  
Justice League of America  
6 4  
Aquaman  
Batman  
Superman  
Spiderman  
The Flash  
The Green Lantern  
Justice League of America Membership: Should Spidey be Eligible?  
2 1  
Yes  
No  
3  
122 5  
2 1 3  
4 1 2  
6 1 5  
8 1 1  
10 1 4  
43 13  
2 1 5  
4 1 2  
6 1 1  
8 1 4  
10 1 3  
21 2 6  
23 2 3  
25 2 2  
27 2 4  
29 2 1  
31 2 5  
70 3 1  
75 3 2  
8365 5  
3 1 1  
6 1 4  
9 1 5  
12 1 2  
15 1 3

*sample input continued on next page*



**2000/2001 ACM SOUTHERN CALIFORNIA REGIONAL  
SCHOLASTIC PROGRAMMING CONTEST**

**Problem 6  
Team Packing**

Every year Swamp County holds its regional collegiate programming contest. One part of running the programming contest is assigning teams to workstations. The organizers attempt to locate teams such that teams from the same school are not assigned workstations close to each other. In the past this has been an ad hoc process. Your team will develop an application to allow the organizers to compare their ad hoc workstation assignments with the best possible layout.

Input to your program is a list of Cartesian coordinates of workstation locations (in integer feet) followed by one line per school giving the number of teams from the school and the school name. There will be a maximum of 14 workstations, and 4 teams per school. The last workstation will always be located at '0,0'. The dimensions of the largest possible workstation lab are 200 by 200 feet. Also, the number of teams will not exceed the number of workstations, and there will always be at least one school with at least two teams.

Output from your program is the *square* of the minimum distance between any two teams from the same school after teams have been assigned workstations that maximize this distance. The output is one line containing the left justified integer value that is the squared distance.

*Sample Input*

```
3,0 6,0 9,0
0,2 3,2 6,2 9,2
0,5 3,5 6,5 9,5 14,3 14,6 0,0
3 Bogy U
3 Gator State
1 Armadillo Tech
2 CFTBL City College
1 Moss Institute
3 SC Polytechnic
```

*Sample Output*