

**ACM Pacific NW Region Programming Contest
10 November 2001**

**PROBLEM A
BEJMUL (JUMBLE)**

Most of you are probably familiar with Jumble™, a daily newspaper feature, where you reconstruct scrambled words. For example, you might be given "ROYIN" and have to come up with "IRONY".

One aspect that makes a particular scrambling of letters "hard" is if it looks like a "real" word. Thus, "ROYIN" is tougher scramble than "NRYIO" because it confuses your brain into thinking that it's already a word.

Another aspect that makes a scrambling harder is whether letters are in their correct position. So, "RIONY" isn't a very good scramble, even if it might kind-of look like a real word.

Your task is to write a program that scores scrambles as "good", "fair", "poor", or "not". Your program will read pairs of words and score them. A scramble is "good" if none of its letters are in the correct place and it looks "real" (more on that later). A scramble is "poor" if it doesn't look "real" and has either the first letter in place or any two consecutive letters in place. If the word isn't scrambled at all, it is said to be "not" scrambled. Otherwise, the scramble is "fair".

How do we know if a word looks "real"? We will use an extremely crude heuristic that the word must alternate between vowels ('Y' is a vowel for these purposes) and consonants. However, certain groups of vowels and consonants are allowed:

AI AY EA EE EO IO OA OO OY YA YO YU
BL BR CH CK CL CR DR FL FR GH GL GR
KL KR KW PF PL PR SC SCH SCR SH SHR
SK SL SM SN SP SQ ST SW TH THR TR
TW WH WR

Also, all double consonants are allowed. No other combinations are allowed, so "SWR" wouldn't be good, even though both "SW" and "WR" are both OK.

Input:

The input for your program will be pairs of words, all upper-case letters, alternating until a "word" of 999 is encountered. The first word of the pair is the original word; the second is the scrambled version that you must evaluate. You may assume that the words are anagrams of one another and contain no spaces. Words will be at least 3 letters in length.

Input will be from the text file **A.in**

Output:

For each pair, you should print how well the second scores as a scramble of the first. Use the format shown in the sample.

(Sample I/O on back)

ACM Pacific NW Region Programming Contest

10 November 2001

Sample I/O:

Input:

```
SPAM
MAPS
IRONY
RIONY
IRONY
ONYRI
IRONY
IRONY
IRONY
INOYR
IRONY
IOYRN
999
```

Output:

```
"MAPS" is a fair scramble of "SPAM"
"RIONY" is a fair scramble of "IRONY"
"ONYRI" is a good scramble of "IRONY"
"IRONY" is not a scramble of "IRONY"
"INOYR" is a fair scramble of "IRONY"
"IOYRN" is a poor scramble of "IRONY"
```

DUCK GOLO! (That's a scramble of 'GOOD LUCK!' ☺)

**ACM Pacific NW Region Programming Contest
10 November 2001**

**PROBLEM B
TALES FROM DECRYPTION**

Alice has designed a secret encryption method she wants Bob to use when he sends ASCII messages to her. The method is based on arithmetic in the integers mod 17.

1. Alice tells Bob to partition his ASCII message (8 bit bytes) into 4-bit nibbles m_1, m_2, m_3, \dots
2. Alice asks Bob to add 1 to each nibble to get integers n_1, n_2, n_3, \dots (so each is from the set $\{1, 2, \dots, 16\}$.)
3. Alice asks Bob to group these into threes, such as $v = (n_1, n_2, n_3) = (14, 5, 7)$, augmented in the last triple by zeros, if necessary, so that each triple is complete.
4. Alice has secretly provided Bob with a sequence of 9 numbers, d_1, d_2, \dots, d_9 , each from the set $\{0, 1, \dots, 16\}$.
5. Alice asks Bob to group the nine numbers into a 3 by 3 matrix thus:

$$A = \begin{matrix} & d_1 & d_2 & d_3 \\ & d_4 & d_5 & d_6 \\ & d_7 & d_8 & d_9 \end{matrix}$$

6. Bob keeps A secret and follows Alice's instructions to use matrix multiplication to compute, for each triple v , $Av = w = (c_1, c_2, c_3)$ reducing mod 17 each entry of w so that these are in the range $0, 1, \dots, 16$. For example, for the matrix A :
$$\begin{matrix} 1 & 2 & 5 \\ 3 & 4 & 1 \\ 1 & 2 & 1 \end{matrix}$$
and the triple v in step 3 the encrypted triple would be $(8, 1, 14)$.
7. Alice knows the matrix A and knows how to decrypt a sequence of triples to get Bob's original ASCII message. (This is not a public key method, since A , supplied by Alice, must be kept secret by Bob.)

Your job is to program this decryption, given the matrix A and the sequence of triples from Bob. (Alice's choice of A is guaranteed to enable a unique solution to the decryption process.) Bob's message starts as printable ASCII characters or spaces (this is guaranteed). You are required to recover and print Bob's ASCII message.

Input:

Input to your program will be a series of data sets. The first line of input will be an integer, denoting the total number of data sets to process. Following this line will be the data sets. Each data set contains the matrix "A" (d_1, d_2, \dots, d_9), an integer, n , (the number of triples in Bob's message), and Bob's set of triples. Each data set is constructed thusly:

```
d1 d2 d3
d4 d5 d6
d7 d8 d9
n
t1 t2 t3 t4 t5 t6...
```

You are guaranteed at least 1 triple, and all triples in the data set will be on a single line.

Input will be from the text file **B.in**

ACM Pacific NW Region Programming Contest

10 November 2001

Output:

Bob's original ASCII message, one line per data set. (The original message will be less than 50 characters.)

Sample I/O:**Input:**

```
2
1 2 5
3 4 1
1 2 1
2
7 8 4 9 16 13
1 2 5
3 4 1
1 2 1
3
7 8 4 9 16 13 16 3 16
```

Output:

```
ME?
ME?M
```

Note that in this example, matrix "A" is the same for each data set, but this will not necessarily be the case!

ACM Pacific NW Region Programming Contest
10 November 2001

PROBLEM C TRIANGLES

Write a program that, given an NxN matrix of characters, determines the number of non-trivial single-character filled "standard" triangles in that matrix.

A "standard" triangle is an isosceles right triangle, with either:

a) the legs aligned along any two dimensions of the matrix, for example:

```
A      BBB
AA     BB
AAA    B
```

b) the hypotenuse aligned along any one dimension of the matrix, for example:

```
      B
     BB
    BBB
   BBB
  BBB
 BBBB
```

(These don't look like right triangles, because the font isn't perfectly square, but they are in terms of the matrix).

No other triangles are counted.

A non-trivial triangle must contain at least 3 letters (a single letter is a trivial triangle).

Input:

The input for your program will be a sequence of matrices. Each matrix will start with a dimension (N) that will be less than twenty, followed by N rows of N upper-case letters. The input ends with a single zero (0) as the dimension.

Input file for this problem is **C.in**

Output:

For each matrix, you should print the total number of non-trivial right triangles in parentheses, followed by the number of non-trivial triangles for each character in the matrix.

Sample I/O:

Input:

```
3
AAB
ABB
BBB
4
AABB
ABBB
BBBB
BBBB
0
```

Output:

```
(10) 1 A 9 B
(51) 1 A 50 B
```

ACM Pacific NW Region Programming Contest
10 November 2001

PROBLEM D
DETOURS!

You are working for the Department of Transportation (DOT) in road repair. When a road is closed for repairs, the public would like the marked detour to be the *shortest* available combination of roads to travel from one end of the closed road to the other.

To simplify the data, we will only consider roads between cities/towns/villages/..., rather than trying to give unique identifiers to crossroads that do not have such names.

After your program has read in the road data set, it will read in a series of town name pairs that mark the roads to be closed. Your program is to find the *shortest* detour between each of the two towns, and specify it by listing in order the towns along the detour - you may use either of the two towns as the starting point - and then list the towns along the path to the ending point. It is understood that there is a road between each pair of towns in your listing. At that point, list also the total length of the detour.

Input:

- The first line contains the number of towns (an integer, N) in the data set. The rest of the first line is to be discarded (there may be additional notation text)
- The next N records contain the names of the towns, each on a separate line. They may or may not contain embedded blanks in their names (for instance, *Coeur d'Alene* and *Wilbur*).
- Subsequent lines contain three items that represent the roads (whitespace delimited):
 1. The town at one end of the road
 2. The town at the other end of the road
 3. The length of the road connecting those two towns.If the name of either town contains whitespace, it will be enclosed in double quote marks. (See sample data)
- The end of road information is marked by this record: `EOD EOD 0`
- Finally, there are a series of town pairs (whitespace delimited). These each represent a road to be closed for repairs, and consequently, two towns between which you must find the shortest path. Again, should the town names contain whitespace, they will be enclosed in double quotes. This portion of the data set is terminated by this town pair: `EOD EOD`

Input file for this problem will be **D.in**

Output:

- On one line, list all of the towns along the detour, including both end-point towns. The list can begin with either town.
- On a separate line following the detour path, give the length of the detour.
- Follow this with a blank line.
- Should your program encounter a city that is not in the list of N cities, either in the list of roads or in the road-closure pairs, display a single line stating "*CityName* is not a recognized town." If the unrecognized city is the first of that record, do not process the rest of the record.
- Follow this with a blank line.
- Should a road-closure pair contain recognized cities, but cities with no direct road between them, display a line stating: "There is no road directly from *CityName1* to *CityName2*." (City names may be in any order in this statement.)
- Follow this with a blank line.

ACM Pacific NW Region Programming Contest

10 November 2001

Sample I/O:

Input:

8
Connell
Coulee City
Davenport
George
Moses Lake
Ritzville
Sprague
Wilbur

Connell "Moses Lake"	46
Connell Ritzville	45
Connell Wilbur	99
"Coulee City" George	55
"Coulee City" "Moses Lake"	52
"Coulee City" Wilbur	35
Davenport Sprague	38
Davenport Wilbur	32
George "Moses Lake"	31
"Moses Lake" Ritzville	42
Ritzville Sprague	23

EOD EOD 0

"Coulee City" "Moses Lake"
George "Moses Lake"
Seattle Spokane

EOD EOD

Output:

Coulee City George Moses Lake
Total distance: 86 miles

Moses Lake Coulee City George
Total distance: 107 miles

Seattle is not a recognized city.

**ACM Pacific NW Region Programming Contest
10 November 2001**

PROBLEM E TAX RELIEF

In June the IRS notified taxpayers that the United States Congress passed and President George "W." Bush signed into law the Economic Growth and Tax Relief Reconciliation Act of 2001. As part of the immediate relief, taxpayers would receive a check, the amount of which was based on Information submitted on their 2000 federal tax return. The notification described how the check amount was calculated, reproduced here in Table 1. Distribution of the checks was based on the schedule of Table 2.

As a new employee at the Department of the Treasury, you have been tasked with developing the program to determine the value of each taxpayer's check and to determine how much is to be sent out each applicable week.

Table 1. Refund Amount

If your filing status is:	Then:
Single or Married Filing Separately	The amount of your check will be the lesser of: \$300, 5% of your taxable income, or your income tax liability
Head of Household	The amount of your check will be the lesser of: \$500, 5% of your taxable income, or your income tax liability
Married Filing Jointly or Qualifying Widow(er)	The amount of your check will be the lesser of: \$600, 5% of your taxable income, or your income tax liability

Table 2. Distribution Dates

If the last two digits of your Social Security Number are:	You should receive your refund the week of:
00 – 09	July 23
10 – 19	July 30
20 - 29	August 6
30 – 39	August 13
40 – 49	August 20
50 – 59	August 27
60 – 69	September 3
70 – 79	September 10
80 – 89	September 17
90 – 99	September 24

Input:

Input to your program is a series of lines, one line per taxpayer, consisting of Social Security Number, filing status, taxable income, and tax liability, each separated by single spaces. The format for the Social Security Number is ###-##-####, where # is a decimal digit, 0--9.

The filing status is a single digit,

- 1 Single
- 2 Married Filing Jointly

ACM Pacific NW Region Programming Contest

10 November 2001

- 3 Married Filing Separately
- 4 Head of Household
- 5 Qualifying Widow(er)

Taxable income and tax liability are whole integer dollars. Input is terminated by a Social Security Number consisting of all zeros.

Program input will be from the text file **E.in**

Output:

For each taxpayer, output a summary line reporting the following:

- Social Security Number (exactly as input)
- two spaces
- a dollar sign (\$) and the check amount, left justified with no leading zeros as: dollars, a decimal point, and two digits of cents

Then, for each week for which checks are to be distributed, output a summary line reporting the following:

- number of checks to be distributed that week (no leading space)
- two spaces
- a dollar sign (\$) and the week's total distributed amount, left justified with no leading zeros, as: dollars, a decimal point and two digits of cents
- one space
- the week exactly as listed in Table 2

List the weeks in the same order as Table 2.

Sample I/O:

Input:

```
123-45-0001 1 40100 8200
890-12-3456 4 98910 24182
123-45-0002 4 6000 904
```

Output:

```
123-45-0001 $300.00
890-12-3456 $500.00
123-45-0002 $300.00
2 $600.00 July 23
1 $500.00 August 27
```

**ACM Pacific NW Region Programming Contest
10 November 2001**

**PROBLEM F
WORM WATCHER**

Swamp County Cable, which has a cable modem ISP operation, wants to get an idea of how often its customers are getting probed or attacked by hackers or worms. To get a handle on this, it has reserved a part of its IP address space, which has never been assigned to anyone, to watch in detail.

Your job is to convert the router accounting summary records for these addresses into an hour-by-hour count of attacks.

Here are some sample accounting records:

```
0830.19:29:57.65      61.121.70.13      2328      64.12.197.2      80      6
171
0830.19:30:40.709    131.91.143.40     3191 64.12.197.145   80      6      3
0830.19:31:21.629    200.56.110.205   3043 64.12.197.189   80      6
1
0830.19:30:40.705    131.91.143.40     3191 64.12.197.145   80      6
168
```

The fields are:

<u>Field Number</u>	<u>Contents</u>	<u>Format</u>
1	date-time	mmdd.hh:mm:sec
2	source IP address	dotted decimal number
3	source port	decimal number [0..65535]
4	destination IP address	dotted decimal number
5	destination port	decimal number [0..65535]
6	protocol	decimal number [0..255]
7	number of packets	decimal number [1..65535]

The fields are separated by one or more blanks.

A dotted decimal number is a 32 bit value, high order byte first, with a decimal representation of each 8-bit byte [0..255], each separated by a period (.). For example, 1.2.255.15 is 0102FF0F in hexadecimal.

The date-time field consists of a 2 digit month [01..12], a 2 digit day [01..31], a period (.), a 2 digit hour [00..23], a colon (:), a 2 digit minute [00-59], a colon (:), and a seconds field [0..59.999].

What is of interest is the number of unique source IP addresses seen in each hour. An hour lasts from, for example, 01:00:0 until 01:59:59.999. Of course during some hours, no entries may appear if there is a major outage or if no attacks occur during that hour. In this case, don't print anything for that hour.

The records are almost sorted by the date-time field. A record will never be more than 5 minutes earlier than the latest record seen before.

From time to time there are attacks with forged source IP addresses that will result in thousands of unique source addresses in any given hour. If the count of unique source addresses exceeds 500 in any given hour, you will just indicate that it is greater than 500, rather than the exact count.

ACM Pacific NW Region Programming Contest

10 November 2001

Input

Input is a series of test cases. Each test case consists of accounting records, each at most 100 characters in length. Each test case is ended by a blank line or end of file. There will never be more than 10 days in a test case.

Input for this problem will be from the text file **F.in**

Output

Output will be one line for each hour, with data from the earliest time to the latest time of any record in the test case. The line has 3 fields: date; hour; count. The date is of the form dd-mmm where dd is the 2 digit day [01..31] with a leading 0 if necessary and mmm is a 3 character lower case abbreviation for the month [jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec]. Put exactly one blank after the date field. The hour is a 2 digit hour [00..23] with a leading 0 if necessary. Put exactly one blank after the hour field. The count field is the count of unique source IP addresses seen during that hour, left adjusted, with no leading 0 or trailing spaces. If the count is greater than 500, print ">500" for the count. Skip one line between test cases.

Sample Input:

```

0830.19:29:57.65      61.121.70.13      2328      62.225.197.2      80      6
171
0830.19:30:40.709    131.91.143.40     3191 62.225.197.145  80      6
3
0830.19:31:21.629    200.56.110.205    3043 62.225.197.189  80      6
1
0830.19:30:40.705    131.91.143.40     3191 62.225.197.145  80      6
168

0830.19:30:40.709    131.91.143.40     3191 62.225.197.145  80      6
3
0830.19:31:21.629    200.56.110.205    3043 62.225.197.189  80      6
1
0830.19:30:40.705    131.91.143.40     3191 62.225.197.145  80      6
168

1231.22:30:40.709    131.91.143.40     3191 62.225.197.145  80      6
3
0101.00:00:00.629    200.56.110.205    3043 62.225.197.189  80      6
1
1231.23:59:40.705    131.91.143.40     3191 62.225.197.145  80      6
168
0101.02:04:00.629    200.56.110.205    3043 62.225.197.19  80      6
1
0101.02:01:05.0      200.56.110.205    3043 62.225.197.18  80      6      1
0101.02:08:13.48     200.14.13.220     3043 62.225.197.189  80      6
1

0723.00:20:26.526    62.14.165.180     2409 62.225.197.204    53     17
19
0723.00:26:53.143    64.174.246.90     3914 62.225.197.72     80      6
3
0723.00:26:53.147    64.174.246.90     3914 62.225.197.72     80      6
171
0723.01:38:00.477    213.174.70.225    0 62.225.197.15    2816    1
1
0723.01:38:00.473    213.174.70.225    0 62.225.197.15    2816    1
118
0723.01:42:15.386    210.77.158.1      0 62.225.197.0     2816    1
1
0723.01:49:58.120    172.185.107.238    1 62.225.197.113    62331    6
20
0723.01:49:58.120    172.185.107.238    1 62.225.197.113    62331    6
1

```

ACM Pacific NW Region Programming Contest

10 November 2001

0723.01:51:02.169	210.184.94.33	1	62.225.197.29	50386	6
1					
0723.01:51:02.173	210.184.94.33	1	62.225.197.29	50386	6
120					
0723.01:56:05.938	213.174.70.225	0	62.225.197.15	2816	1
118					
0723.01:56:05.938	213.174.70.225	0	62.225.197.15	2816	1
1					
0723.02:03:57.652	172.176.163.61	1	62.225.197.33	19146	6
1					
0723.02:03:57.656	172.176.163.61	1	62.225.197.33	19146	6
19					
0723.02:25:35.771	172.185.107.238	1	62.225.197.111	21414	6
19					

Sample Output:

30-aug 19 3

30-aug 19 2

31-dec 22 1

31-dec 23 1

01-jan 00 1

01-jan 02 2

23-jul 00 2

23-jul 01 4

23-jul 02 2

**ACM Pacific NW Region Programming Contest
10 November 2001**

**PROBLEM G
WHAT'S MY SIZE?**

When designing an HTML page for display on the World Wide Web, the length of time it takes to display the page in the user's browser is a major consideration. Studies show that most users will not wait more than 10 seconds for a page to download before "clicking out", and costing the web site valuable "eyeballs".

Many current HTML editors have the ability to let the designer know page download time for the "average" user. Your task is to write such a utility to compute download time, assuming a "static" HTML document. (For our purposes, a static HTML document is one that contains no multimedia, Java applets or dynamically generated content. It consists of only HTML code and images.) Download time will be determined by the total size (in bytes) of all images to be displayed in the document, plus the document size (in bytes, 1 character = 1 byte), which is computed by its character count. This total number of bytes is used to determine the download time, assuming 1000 bytes = 0.2 second.

For example, given a simple HTML document such as this:

```
<HTML>
<HEAD>
<TITLE>ACM</TITLE>
</HEAD>
<BODY>
<IMG SRC="acm.jpg"><BR>
<IMG SRC="contest.jpg">
</BODY>
</HTML>
```

and assuming the size of "acm.jpg" is 10k bytes and "contest.jpg" is 5k bytes we can compute the download time as follows:

- Number of bytes in document: 112 (assume there is a CR/LF character at the end of EACH line)
- Number of bytes in images: 15000
- TOTAL SIZE: 15112 bytes
- DOWNLOAD TIME: 3.02 seconds

For this problem, we are going to add an attribute to the IMG tag, called "size". This attribute will have an integer value, denoting the image file size in bytes. For example, the sample above would look like this:

```
<HTML>
<HEAD>
<TITLE>ACM</TITLE>
</HEAD>
<BODY>
<IMG SRC="acm.jpg" size="10000"><BR>
<IMG SRC="contest.jpg" size="5000">
</BODY>
</HTML>
```

ACM Pacific NW Region Programming Contest

10 November 2001

Note that if an image is used more than once on the page, it is only counted once in determining download time, since it won't have to be downloaded again, once it's been downloaded originally. (Images with the same file name, but in different directories are NOT the same image. Image names ARE the same if a case-insensitive string compare declares them to be equal.)

Note that HTML tags are NOT case sensitive.

Note that other HTML tags besides the tag can contain the "SRC=" attribute and the "size=" attribute.

Note also that an tag can contain the "SRC=" attribute along with other attributes, and these can be in any order. For instance, for our purposes, all of these are valid tags:

```
<IMG SRC="acm.jpg" size="10000">  
<img BORDER="0" SRC="/images/contest.gif" size="5000">  
<Img HEIGHT="42" WIDTH="42" size="5000"  
src="../Images/Answers/Answer.jpg">
```

(You may assume that the path/image name and size value are surrounded by double quotes.)

Specifically, you may assume that the only spaces in a tag will be either between name=value pairs or embedded in quoted strings. More specifically, there will be no spaces between the < and tag name, around the equal signs, before the >, or anywhere else not already specified.

Input:

Input for this problem will begin with a single integer on a line, denoting the number of HTML files to process. Following this line will be a series of document listings. Each HTML document listing will begin with <HTML> tag and end with </HTML> tag (not case-sensitive!), and will contain 0 or more image tags. There will be no blank lines within the HTML document. The ending </HTML> tag for each test document listing will be the last item on a line. Image file names (including the path) will not exceed 80 characters.

Input for this program will be the text file **G.in**

Output:

Output consists of a series of lines, one per HTML document, listing the time, in seconds, to download that document, rounded to 2 decimal places.

ACM Pacific NW Region Programming Contest

10 November 2001

Sample I/O:

Input:

```
2
<HTML>
<HEAD>
<TITLE>ACM Contest</TITLE>
</HEAD>
<BODY>
<HR><H3>IBM Sponsors ACM-ICPC!</h3>
<IMG SRC="acm.jpg" size="5000"><BR>
<p>A new paragraph</p>
<center><img BORDER="0" size="1000" SRC="/images/contest.gif"></center>
&nbsp;
</BODY>
</html>
<HTML>
<HEAD>
<TITLE>Pacific NW Region</TITLE>
<script language="JavaScript" src="../scripts/utils.js"></script>
</HEAD>
<BODY>
<b>PacNW Rocks!</b><br>
<hr>
<table>
<tr>
<td><IMG size="5000" SRC="../IBM.jpg"></td>
<td><p>A new paragraph</p></td>
<td align=center><img BORDER="0" SRC="/images/pacNW.jpg" size="7000">
</td></tr></table>
<br><br>
<br>
<h1>Bye!</h1></BODY></HTML>
```

Output:

```
11.26 seconds
7.50 seconds
```

**ACM Pacific NW Region Programming Contest
10 November 2001**

**PROBLEM H
THE “PAREN” THESIS**

Jane Paren, was sitting around the house one day thinking about a multiplication machine she wanted to design. Her machine would take an ordered sequence of variables or numbers, like this:

$X_1, X_2, X_3, X_4, \dots, X_N$

And then determine all possible ordered, multiplicative “groupings” of the objects like this:

$(X_1 (X_2 (X_3 (X_4 (\dots (X_{N-1} * X_N) \dots))))$
:
:
:
 $((\dots ((X_1 * X_2) X_3) X_4) \dots) X_{N-1} X_N$

Jane hypothesized that a machine could produce these different groupings by repeatedly dividing the main sequence into two parts starting on the left and working toward the right, and then treating each of these sub-sequences just like the main sequence was treated. The very first division into two sub-sequences is shown here:

$((X_1) (X_2 X_3 \dots X_{N-1} X_N))$

Later in the sequence another division of the main sequence into two sub-sequences is shown below:

$((X_1 X_2 X_3 \dots X_{j-1}) (X_j \dots X_{N-1} X_N))$

Both the left and right sequences must be subdivided from left to right just like the main sequence. You will help Jane Paren verify her hypothesis by writing a program that can group a sequence into a series of multiplications using parentheses. Your program will not perform the actual multiplications; it will only find and output all possible ordered parenthesized groupings in the order specified by Jane’s hypothesis.

Input

The input will consist of multiple lines of variable names or numbers.

Each variable name or number will be separated from its neighbors by spaces. The input line will be terminated by carriage-return/line-feed (standard end-of-line marker). Ignore multiple blank lines in the input and treat them as a single blank line.

The program will stop reading input when it reaches the end of the file.

Input for this problem will be from the text file **H.in**

Output

For each input line, the program will generate all possible multiplicative groupings, keeping the numbers or variables from the input line in their original order. **The order in which the multiplicative groupings are output does matter!**

Subsequences consisting of just one element will not have its’ own set of parenthesis.

Subsequences consisting of just two elements are separated with the multiplication symbol * and have a set of parenthesis around them in the output, e.g. $(x*y)$.

For each element of the set of outputs calculated from a given input line, the output will display the line from which the group originated and the number within the group.

Outputs must conform to the same formatting as the sample output, including spaces!

Separate each set of output groupings by a blank line.

ACM Pacific NW Region Programming Contest

10 November 2001

Sample I/O

Input:

```
1 2
North South East West
```

Output:

```
Input Line 1 Group 1. (1*2)

Input Line 2 Group 1. (North(South(East*West)))
Input Line 2 Group 2. (North((South*East)West))
Input Line 2 Group 3. ((North*South)(East*West))
Input Line 2 Group 4. ((North(South*East))West)
Input Line 2 Group 5. (((North*South)East)West)
```