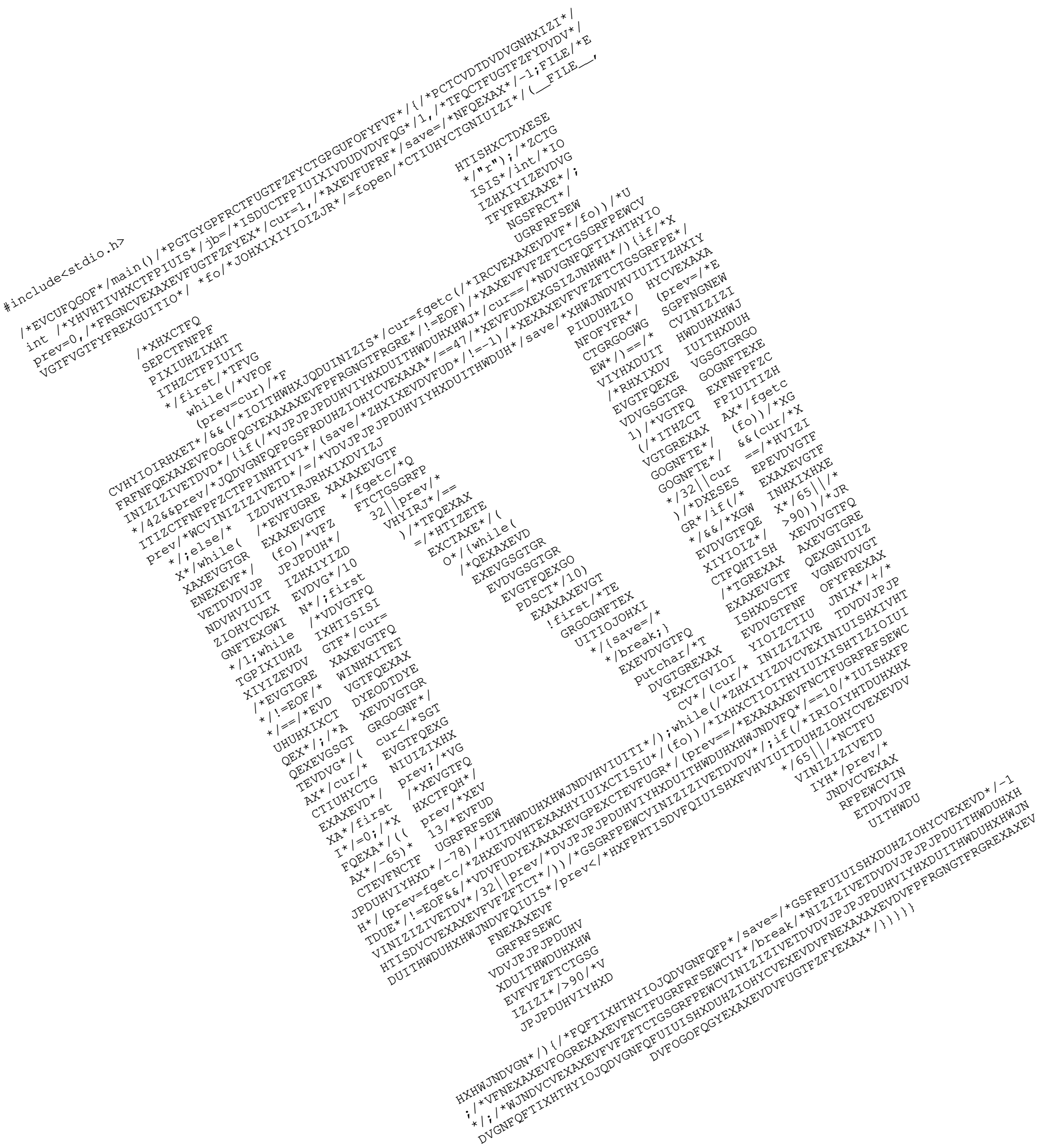
ACM East Central Region
1996 Regional Programming Contest

October 26, 1996

University of Notre Dame

Problem Set



Instructions

There are 7 problems, given in no particular order of difficulty. Please notify a judge immediately if you do not have all 7 problems. There are 15 labeled pages in this handout.

Getting Help

If you have a problem with your computer or computing environment, please notify a student consultant immediately.

Judges will NOT give out helpful hints on any of the programs. We will clarify any questions that you have about the problem, but no help will be given. Please see the handout entitled "Official Rules" for a further explanation.

Input and Output

For programming purposes, you may assume that all data will be of the format specified in the problem specification. That is, no invalid data will be used to judge your program.

Unless otherwise specified in the instructions for a particular problem, you should output numerical values in the natural format for the given data type in the programming language used. I.e., do not attempt to specially format output data that you are not asked to specially format.

You should use double precision floating point for real numbers. By default, the `real` data type corresponds to double precision in the Solaris Pascal compiler in this computing environment.

Examples

The example output in each problem is assumed to have been produced by a C program. C++ and Pascal output may look slightly different because these languages may have different default formats for various data types.

1 Matching Meetings

"Argh! I can't stand it anymore!" exclaimed Dan Hanson, CS Department administrative assistant, as he slammed down the phone.

"What's the matter, Dan?" asked Valentine McKee, who happened to be in the office making some copies.

"Professors and scheduling. I'm trying to schedule this SSC meeting but as soon as I pick a time and start making calls to confirm it, one of the committee members has a conflict. I feel like this is going to go on forever!" He Said.

"Maybe the computer can help," suggested Valentine. "This is the computer science department, after all," she Said.

1.1 Input

The first line of input will be the "current" date.

All dates will be specified in the form: `dayname month date`, where:

- `dayname` is single character from the set { `M, T, W, R, F` } representing Monday, Tuesday, Wednesday, Thursday, and Friday, respectively.
- `month` is an integer such that 1 ≤ `month` ≤ 12.
- `date` is an integer such that 1 ≤ `date` ≤ 31.

The next line of input will contain two integers, n and t. n specifies how many meetings should be scheduled, while t indicates how long each meeting will last. t will be specified in increments of 15 minutes.

All times will be specified as zero padded 4 digit integers in military time. The first two digits indicate the hour (00 corresponds to midnight). The appointment times in the input will be in the range of 09 ≤ `hour` ≤ 17. The last two digits indicate the minute, and will be in the be from the set { `00, 15, 30, 45` } .

You may assume that `date`, while formally bounded by 31, will not be greater than the number of days in the `month`. You may also assume that all years will *not* be leap years.

The rest of the input will contain the schedules for at most 100 individuals. Each schedule will begin with a line specifying the individual's name, followed by a list of appointments for that individual, ending with the word "`done`" on a line by itself. No individual will have more than 100 appointments scheduled. Each appointment will be a date followed by a start time and an end time. No appointment will extend before 9am or after 5pm.

The last line of input will be the word "`done`" on a line by itself.

No appointment will be before the current date, or more than 1 year after the current date.

1.2 Output

Your program should output the first n possible meeting times when all the people from the input can meet for the specified amount of time (t). Each line of output should contain one meeting date and time. Once a common meeting time is found, it should be considered allocated for all people, and cannot be used to schedule other meetings.

The meetings should be sorted by date and time, with the earliest meeting first.

If less than n meeting times are available, print all possible meeting times followed by "`No more times available`".

1.3 Example

The following is sample input for this problem.

```
M 8 21
2 60
Jack Casey
M 8 21 0900 1015
done
Jack Ross
M 8 21 1000 1100
M 8 21 1200 1700
done
Jack Swigert
M 8 21 1600 1700
T 8 22 0900 1000
done
done
```

The following is the corresponding output for the input above.

```
M 8 21 1100
T 8 22 1000
```

2 Supercomputer Selection, The Sequel

Valentine sighed as she sat down to lunch with her cousin, Martin Thiel, in the BIT student union Diner.

"What's the matter, Valentine?" Martin asked.

"My presentation to the SSC. They all really liked the graphical displays of the criteria and the selection process," Valentine grudgingly admitted. "But, Prof. Everett pointed out that the criteria would change over time and since we will be using the machine over several years, that we needed to take that into account as well."

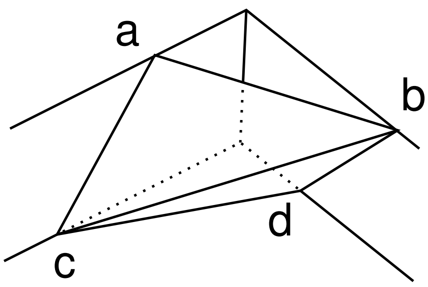
"Maybe you could make separate graphs for each year?" suggested Martin.

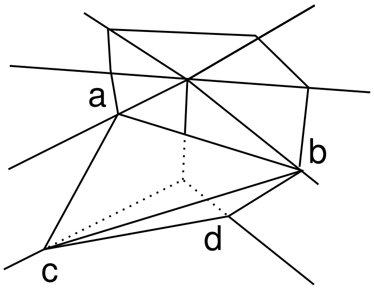
"That's what I said to Prof. Everett, but she didn't think that would be sufficient — it would only pick the best machine for a particular year."

As Valentine silently pondered her problem, Jake Briggs, one of Valentine's CS classmates, joined Martin and Valentine at their table. "What's up?" he asked. Valentine explained her dilemma.

"Simple, " Jake mumbled through bites of his club sandwich. "Make three-dimensional shapes, using time as the third axis. The computer with the largest volume wins."

Valentine sketched out a perspective drawing of such a volume as well as a detail of a single quadrant. "What do we do here at the edges of the axes? There are four points." Valentine pointed to the points a, b, c, and d in her drawing of the quadrant.





"Just use two planes, one determined by a, b, and c, and the other determined by b, and c, and d."

"Whoa! Problem!" exclaimed Martin. "Those volume things are way too complicated for slideware. You'll just confuse everyone."

"No problem," Jake said. "We can use our virtual reality cave over in the JFK building. It even has tactile feedback gloves. The committee members can touch the volumes if they want."

2.1 Input

The first line of input will contain two integers. The first integer, n, specifies the number of criteria that will be measured for each year. This value will be such that $1 \le n \le 100$. The second integer, y specifies how many years will be considered. The value of y will be such that $1 \le y \le 10$.

The rest of the input will consist of sequences of possible configurations of supercomputers; each line will contain n floating point values. The supercomputer configurations will be grouped in sets of y, i.e., the first y lines (after the initial line of two integers) will be the yearly configurations for the first computer. The next y lines will be the yearly configurations for the second computer, etc.

The first number on the input line corresponds to the value on the first axis, the second number corresponds to the value on the second axis, and so on.

The final line of input will contain n floating point zeros.

2.2 Output

The output should be the configuration number (indexed from 1) which gives the configuration with the greatest volume, along with the value of the volume. The volume is to be given to two decimal places. Assume that the polygons are spaced with unit time along the time axis.

In case of a tie (i.e., multiple configurations are given that produce a polygon of equal volume), output the one with the lowest configuration number.

2.3 Example

The following is sample input for this problem.

```
5 2
1.2 2.3 3.4 4.5 5.6
0.2 3.3 4.4 5.6 4.5
10.0 11.0 12.0 13.0 14.0
9.0 12.0 11.0 14.0 13.0
0.0 0.0 0.0 0.0 0.0
```

The following is the corresponding output for the input above.

```
2 339.84
```

3 Polygonal Puzzle

To help model the three dimensional volumes that describe each supercomputer choice over time, Valentine McKee and Jake Briggs decided to construct some cardboard mock-ups. Valentine began by cutting out polygons of each of the supercomputer criteria for each year. As she was bringing the cut-outs to Jake's lab, where they intended to turn the flat polygons into the actual volumes, Valentine bumped into Ren McCormack. Literally. And she dropped all of her polygon cut-outs onto the ground.

"Ren! Why don't you look where you're going," Valentine sighed. "I dropped all of my polygons. Now I'm going to have to start all over cutting these out." Valentine had labeled the polygons prior to cutting them out, but unfortunately, the labeling was not part of the actual polygon and was lost when they were cut out. "I don't know which is which now."

"Sorry." Ren looked downcast, then brightened. "Hey, do you want to go to the dance with me this weekend?"

"I don't think so, Ren," replied Valentine, "I think I'll be cutting out and sorting polygons all weekend." Valentine stooped and began picking up the cut-outs. Ren was a great dancer but he also tended to show it off. The one time that Valentine had gone to a dance with him, Ren had completely embarrassed her with his antics.

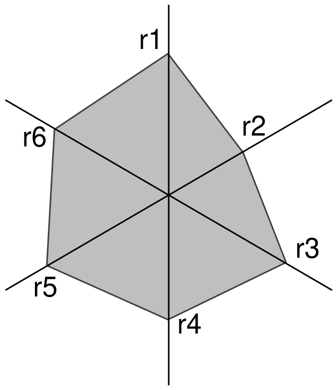
"Can't you use a computer or something to help you sort those things out?" Ren picked up one of the polygons and squinted at it. (Ren was a theater major.)

Valentine thought for a moment. "Nope," she said. "I'm just going to have to start over. Have fun at the dance this weekend, though." To herself, she thought, "Yes, I can use a computer. The only other thing I need is that camera that Jake already has in the lab to help me see the Big Picture."

3.1 Input

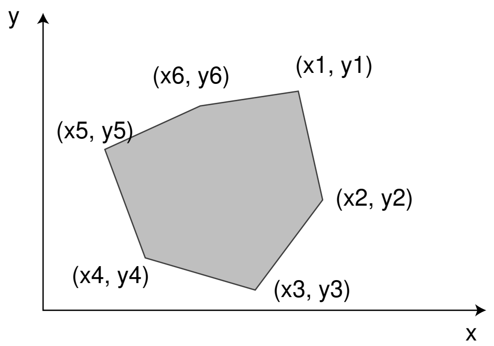
The first line of input will consist of two integers separated by a space. The first integer specifies the number of polygons to be sorted. The second integer specifies the number of vertices for each polygon.

The rest of the input will consist of two lists of coordinates. The first is a list of coordinates describing the shape of each polygon in terms of the values on each of its axes (i.e., the values for `r1` through `r6` in the figure below).



The vertices of each polygon are distributed at equal angles around the origin, with the first vertex aligned at 12 o'clock. The vertices are given in clockwise order. The polygons in this list are unique with respect to rotation and dilation. That is, no polygon will be exactly like any other polygon (having the same vertices in the same locations), nor can any polygon be rotated about its center and/or uniformly scaled such that it will be exactly like another polygon in the list.

The second list of polygon coordinates corresponds to the polygons in the first list, but as measured by a camera. The vertices of each polygon in this case will be given as (x, y) pairs with respect to the camera's viewpoint origin and are given in clockwise order. The first vertex given is arbitrary, however.



Note that the camera's view of each polygon may include magnification or de-magnification (which may be different for each viewed polygon). Moreover, the camera can only resolve the polygon vertices to within plus or minus one pixel in its field of view, so there may be some jitter associated with the viewed vertex locations. For each polygon, the maximum amount of jitter at each vertex is guaranteed to be less than one percent of the distance from the origin to the vertex farthest away from the origin.

All of the polygons viewed by the camera will be right-side up.

3.2 Output

The output is a list of instructions indicating how to rearrange the scrambled polygons. That is, for each polygon in the second list, the output should indicate the polygon to which it corresponds in the first list, the amount by which it must be rotated in a clockwise direction to align the vertices. Thus, one line of output consists of an integer index i (indexing starts from 1), a floating point angle θ (with $0 \leq \theta < 360$) given to the nearest tenth of a degree.

3.3 Example

The following is sample input for this problem.

```
2 4
1 1 2 1
1 1 1 1
1 3 2 3 2 2 1 2
2.5 3.5 2 2 1 2 1 3
```

The following is the corresponding output for the input above.

```
2 45.0
1 135.0
```

4 Word-Search Wonder

The Pyrates Restaurant was starting to fill up as Valentine McKee walked in. She scanned the crowd for her sister, brother-in-law, and nephew. Seeing her sister waving from the far end of the restaurant, she made her way back to their booth. “Hi, Valentine,” her sister and brother-in-law, Niki and Dennis Chapman, greeted her.

“Hi, guys,” she replied. “What are you doing, Wade?” she asked her nephew. He was busy working on one of the restaurant’s activity sheets with a crayon.

“I’m doing a word search game,” Wade explained. “I have to find all of these words in this big mess of letters. This is really hard.” Wade looked intently at the paper in front of him.

“Can I help?” asked Valentine, looking across the table at the activity sheet.

“Sure. These are the words we’re looking for. They’re the names of different kinds of Planes, Trains, and Automobiles.”

4.1 Input

The first line of input will specify the length (in characters) of the sides of the letter matrix (the matrix of letters will be square). The length, l, will be in the range $1 \le l \le 100$. The next l lines of input will be the matrix itself, each line will contain l uppercase letters.

A list of words will follow. Each word will be on a line by itself; there will be 100 or fewer words. Each word will be 100 or fewer characters long, and will only contain uppercase letters.

The final line of input will contain a single zero character.

4.2 Output

Your program should attempt to find each word from the word list in the puzzle. A word is “found” if all the characters in the word can be traced in a single (unidirectional) horizontal, vertical, or diagonal line in the letter matrix. Words may not “wrap around” rows or columns, but horizontal and diagonal words may proceed from right to left (“backwards”). For each word that is found, your program should print the coordinates of its first and last letters in the matrix on a single line, separated by a single space. Coordinates are pairs of comma-separated integers (indexed from 1), where the first integer specifies the row number and the second integer specifies the column number.

If a word is not found, the string “`Not found`” should be output instead of a pair of coordinates.

Each word from the input can be “found” at most once in the puzzle.

4.3 Example

The following is sample input for this problem.

```
5
EDEEE
DISKE
ESEEE
ECEEE
EEEEE
DISC
DISK
DISP
0
```

The following is the corresponding output for the input above.

```
1,2 4,2
2,1 2,4
Not found
```

5 MPI Maelstrom

BIT has recently taken delivery of their new supercomputer, a 32 processor Apollo Odyssey distributed shared memory machine with a hierarchical communication subsystem. Valentine McKee's research advisor, Jack Swigert, has asked her to benchmark the new system.

"Since the Apollo is a distributed shared memory machine, memory access and communication times are not uniform," Valentine told Swigert. "Communication is fast between processors that share the same memory subsystem, but it is slower between processors that are not on the same subsystem. Communication between the Apollo and machines in our lab is slower yet."

"How is Apollo's port of the Message Passing Interface (MPI) working out?" Swigert asked.

"Not so well," Valentine replied. "To do a broadcast of a message from one processor to all the other $n-1$ processors, they just do a sequence of $n-1$ sends. That really serializes things and kills the performance."

"Is there anything you can do to fix that?"

"Yes," smiled Valentine. "There is. Once the first processor has sent the message to another, those two can then send messages to two other hosts at the same time. Then there will be four hosts that can send, and so on."

"Ah, so you can do the broadcast as a binary tree!"

"Not really a binary tree — there are some particular features of our network that we should exploit. The interface cards we have allow each processor to simultaneously send messages to any number of the other processors connected to it. However, the messages don't necessarily arrive at the destinations at the same time — there is a communication cost involved. In general, we need to take into account the communication costs for each link in our network topologies and plan accordingly to minimize the total time required to do a broadcast."

5.1 Input

The input will describe the topology of a network connecting n processors. The first line of the input will be n, the number of processors, such that $1 \le n \le 100$.

The rest of the input defines an adjacency matrix, A. The adjacency matrix is square and of size $n \times n$. Each of its entries will be either an integer or the character x. The value of $A_{(i,j)}$ indicates the expense of sending a message directly from node i to node j. A value of x for $A_{(i,j)}$ indicates that a message cannot be sent directly from node i to node j.

Note that for a node to send a message to itself does not require network communication, so $A_{(i,i)} = 0$ for $1 \le i \le n$. Also, you may assume that the network is undirected (messages can go in either direction with equal overhead), so that $A_{(i,j)} = A_{(j,i)}$. Thus only the entries on the (strictly) lower triangular portion of A will be supplied.

The input to your program will be the lower triangular section of A. That is, the second line of input will contain one entry, $A_{(2,1)}$. The next line will contain two entries, $A_{(3,1)}$ and $A_{(3,2)}$, and so on.

5.2 Output

Your program should output the minimum communication time required to broadcast a message from the first processor to all the other processors.

5.3 Example

The following is sample input for this problem.

```
5
50
30 5
100 20 50
10 x x 10
```

The following is the corresponding output for the input above.

```
35
```

6 Integer Inquiry

One of the first users of BIT's new supercomputer was Chip Diller. He extended his exploration of powers of 3 to go from 0 to 333 and he explored taking various sums of those numbers.

"This supercomputer is great," remarked Chip. "I only wish Timothy were here to see these results." (Chip moved to a new apartment, once one became available on the third floor of the Lemon Sky apartments on Third Street.)

6.1 Input

The input will consist of at most 100 lines of text, each of which contains a single VeryLongInteger. Each VeryLongInteger will be 100 or fewer characters in length, and will only contain digits (no VeryLongInteger will be negative).

The final input line will contain a single zero on a line by itself.

6.2 Output

Your program should output the sum of the VeryLongIntegers given in the input.

6.3 Example

The following is sample input for this problem.

```
123456789012345678901234567890
123456789012345678901234567890
123456789012345678901234567890
0
```

The following is the corresponding output for the input above.

```
370370367037037036703703703670
```

7 Enigmatic Encryption

One Saturday morning, Valentine McKee attempted to log into `kevin.bit.edu`, the new BIT supercomputer, to work on her new MPI implementation. After a few failed login attempts, Valentine put her head in her hands. “I can’t believe it! I’ve forgotten my password!”

Valentine thought for a few minutes. She remembered choosing her password as a combination of two words from the body of her thesis. The password program on `kevin` had required her to make the password between 5 and 8 characters long with at least one “special” character (non-alpha), so Valentine had also incorporated a numeral into the password. “Either 0, 2, 4, or 8, between the two words,” she recalled. Also, to save typing effort, the she had used only lowercase for the letters. She could not recall, however, whether or not the original words had been all lowercase. She did recall that both words were longer than just a single character.

Valentine started to read through the most recent printout of her thesis, trying to jog her memory of the password, but to no avail. Finally, she said, “My thesis is on line in my lab and I have a record there of my encrypted password. I never thought I would have to use it, though.”

She sighed as she invoked `man crypt` to determine how `crypt()` processes passwords.

crypt(3C) C Library Functions crypt(3C)

NAME

crypt − generate encryption

SYNOPSIS

char ∗**crypt(const char** ∗*key***, const char** ∗*salt***);**

DESCRIPTION

crypt( ) is the password encryption function. It is based on a one-way encryption algorithm with variations intended (among other things) to frustrate use of hardware implementations of a key search.

key is the input string to encrypt, for instance, a user’s typed password. Only the first eight characters are used; the rest are ignored. *salt* is a two-character string chosen from the set [***a-zA-Z0-9./***]; this string is used to perturb the hashing algorithm in one of 4096 different ways, after which the input string is used as the key to repeatedly encrypt a constant string. The returned value points to the encrypted input string. The first two characters of the return value are the *salt* itself.

NOTES

The return value for **crypt( )** points to static data that are overwritten by each call.

The **crypt( )** function is provided for you. You do not have to write it yourself.

ApollOS 1.3 Last change: 17 Apr 1970 1

For Pascal Programmers

Include the following in your program so that you can properly call the `pcrypt()` procedure (a wrapper to the C `crypt()` function). The `pcrypt()` procedure is a library procedure that we will supply — you do not have to write it.

```
type
   key_type     = array[1..8] of char;
   salt_type    = array[1..2] of char;
   encrypt_type = array[1..32] of char;

procedure pcrypt(     key    : key_type;
                      salt   : salt_type;
                  var result : encrypt_type); external c;
```

7.1 Input

The first line of input is the encrypted password. The next lines of input are the contents of Valentine's thesis. Each input line will consist of 80 or fewer characters of printable ASCII text and will be terminated by a newline. No lines will be empty. Words are assumed to be any contiguous set of alphabetic characters (upper or lower case), separated by whitespace or punctuation.

7.2 Output

The output of the program is Valentine's original password.

7.3 Example

The following is sample input for this problem.

```
h8E6dqt5lkL9o
The parallel algorithms were executed on the Connection Machine model
CM-2 --- a single-instruction multiple data (SIMD) parallel computer
which, in its largest configuration, contains 65,536 bit-serial
processors and 2048 Weitek floating-point units (FPU's).
The bit-serial processors are clustered together into groups of 16
within a single integrated circuit, and these IC's are connected
together in a 12-dimensional hypercube.  Two IC's, or 32 processors,
share a single Weitek FPU.  Note that a fully configured CM-2 contains
2048 times as much floating point hardware as a conventional computer
containing a single Weitek FPU (e.g., a SUN-4).
```

The following is the corresponding output for the input above.

```
bit0note
```