Consortium for Computing Sciences in Colleges Northeastern Region
2004 Undergraduate Programming Contest

Union College, Schenectady, NY
Friday, April 23, 2004

Last Minute Reminders

1. The contest ends at noon. No submissions will be accepted after that.

2. Any printouts need your school/team name at the top as a comment. You may not pick up your own printouts. They will be delivered to you.

3. Don't forget to name your input files correctly.

4. For each problem, you may assume that the input is both legal and properly formatted. Hence, no error checking is required.

GOOD LUCK!

1. Memory Tester

You are developing a "memory tester". You show someone a random sequence of capital letters, say

```
A F K Z M L B Q R C J
```

and give someone a short time to view them. You then ask them to enter the correct sequence as input to the program you will write. Your program should accept multiple guesses. For each guess, it should report on the person's recall with the following three measures:

1. The number of letters entered that appeared somewhere in the list
2. The number of letters entered in their proper positions
3. An 'analog' score that smoothly penalizes out-of-position guesses. For each letter that is in the list, add 1 to the score if it is in the correct position, ½ if it is off by one, 1/3 if off by two, and so on.

Caveats:

- Guesses contain the same number of characters as in the original sequence.
- Neither the original sequence nor the guesses will contain duplicate characters.

Input file format:
The first line will contain the number of letters in the sequence. The second line will contain the sequence. The letters will be separated by spaces. The third line will contain the number of guesses. Each line thereafter will contain a guess sequence. An input file for the above example would be:

```
11
A F K Z M L B Q R C J
2
F K A Z M L D Q R J C
A F K Z M L B Q R C J
```

The output must contain the guess sequence and the values computed for the three measures for the first guess, followed by sequence and the three measures for the second guess, and so on. Each measure should be labeled. The labels must include the number of the measure in the list above. All decimal results should be printed to the precision computed. Do not round them. An example output for the input above would be:

```
Guess: F K A Z M L D Q R J C
1. The number of letters that appeared somewhere in the list: 10
2. The number of letters in their proper position: 5
3. The 'analog' score: 7.333333333333333

Guess: A F K Z M L B Q R C J
1. The number of letters that appeared somewhere in the list: 11
2. The number of letters in their proper position: 11
3. The 'analog' score: 11.0
```

2. Star map

You work for the Jet Propulsion Laboratory. They want you to write a program that will take an array containing the digitized representation of a picture of the night sky and locate the stars in it. Each element of the array represents the amount of light hitting that portion of the image when the picture was taken. Intensities range from 0 to 20.

Example intensity array:

```
0  3   4  0   0  0  6   8
5  13  6  0   0  0  2   5
2  6   2  7   3  0  10  0
0  0   4  15  4  1  6   0
0  0   7  12  6  9  10  4
5  0   6  10  6  4  8   0
```

A star is probably located in position A[i,j] of input array A if the following is true:

(A[i,j] + sum of the p neighboring intensities)/(p+1) > 6.0

where a neighbor of position A[i,j] is defined as a horizontally or vertically adjacent element (not diagonally). Thus a corner element has p=2 neighbors, while an interior element has p=4 neighbors.

The desired output is a star map containing asterisks where you have found a star and blanks elsewhere. The output for the example above would be:

```
   0  1  2  3  4  5  6  7
0                     *
1     *
2
3           *
4           *  *     *
5           *  *  *
```

Input file format:
The first two lines will be the dimensions of the intensity array (number of rows, then number of columns). The remaining lines will be the intensities themselves. Each row of intensities in the input file corresponds to a row in the array. There is one space separating each intensity from the next one in a row. The input file for the above example would thus be:

```
6
8
0  3   4  0   0  0  6   8
5  13  6  0   0  0  2   5
2  6   2  7   3  0  10  0
0  0   4  15  4  1  6   0
0  0   7  12  6  9  10  4
5  0   6  10  6  4  8   0
```

The output should contain row and column labels as the example output does.

3. Monotonic Subsequence

A fascinating theorem from analysis states that, given any sequence of n distinct numbers, a monotonic (either increasing or decreasing), but not necessarily contiguous, subsequence of at least $n^{1/2}$ can be made from its elements. For example, the 9 term sequence:

2 -1 3 -7 -4 9 -10 0 8

contains the monotonic subsequences 2, 3, 9 and 2, -1, -7, -10, both of at least length 3.

Write a program that reads input sequences and then finds and prints the longest monotonic subsequence, either increasing or decreasing. If there are several maximum length sequences, print any one of them.

The input file has the following format:

<number of sequences N>
<number of values in first sequence>
<first sequence>
<number of values in second sequence>
<second sequence>
…
<number of values in Nth sequence>
<Nth sequence>

Each sequence will thus be given in two lines, the first with the number of values, the second with the sequence itself. Numbers in sequences will be separated by spaces. An example input file is:

```
2
9
2 -1 3 -7 -4 9 -10 0 8
4
3 2 4 9
```

The output should include the sequence values and the longest monotonic sequence, clearly labeled. The output for the above example would be:

```
Input sequence: 2, -1, 3, -7, -4, 9, -10, 0, 8
Longest monotonic subsequence:  2, -1, -7, -10

Input sequence: 3, 2, 4, 9
Longest monotonic subsequence:  3, 4, 9
```

4. Regular expression search

Write a program that will take two inputs: a single sentence S such as

The quick brown fox jumped over the lazy dog, may jump over the black sheep, and is jumping still.

and a search term T such as

jumped

Your program will determine if T occurs in S. The search will be case-sensitive. T can contain special symbols, as listed in the following table.

Symbol	Stands for	Example	What it matches in example above
*	0 or more characters	jump*	jump jumped jumping
?	Exactly 1 character	?he	The the

The following constraints apply:

- T may contain only alphanumeric characters and the special symbols (no spaces or other punctuation marks). Thus a match will always be a single word.
- S may not contain the symbols * or ?.
- T may contain multiple special symbols.

Here are two more examples:

Example	What it matches in example above
*a**y	may lazy
?he*	The the sheep

Input file format:
The first line will contain S on a single line. The second line contains the number of search terms T, followed by the search terms, one per line.

Example input file:

The quick brown fox jumped over the lazy dog, may jump over the black sheep, and is jumping still.
2
?he*
over

For each search term T in the input file, the output will reprint T followed by the words in S that matched it. The output for the above example input file would be

Search term: ?he*
The
the
sheep

Search term: over
over

A blank line should appear between each set of results, as shown above. If a match for a search term appears more than once in S (such as **over** in the example), it should appear just once in the output.

5. Memory allocation

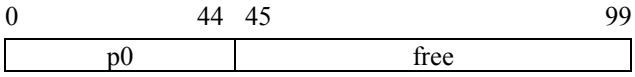
Write a program that simulates memory allocation and deallocation to and from various processes. You start with a block of free memory, M units long, indexed from memory address 0 to address M-1. Allocation is done with the following command

A <name of process> <memory request>

Process names will always have the form p*k* where *k* is a unique process number. For example,

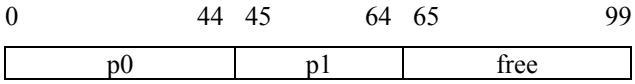
```
A p0 45
```

indicates that process p0 is requesting 45 units of memory. If the original free memory were M=100 units, then after the allocation, memory would look like this:



And after
```
A p1 20
```

memory looks like this:

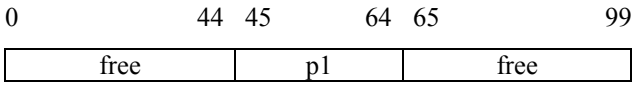


Deallocating memory is done with the command

D <name of process>

For example,
```
D p0
```

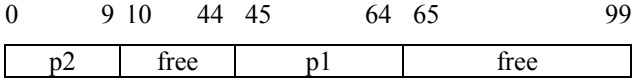
will free up memory currently used by process p0:



If a process request can be filled by more than one free "chunk" of memory, such as

```
A p2 10
```

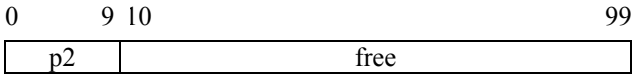
which can fit either before or after p1 in the picture above, then **worst-fit strategy** is used. That is, the biggest "chunk" will be used:



Whenever a deallocated "chunk" is adjacent to another section of free memory, then those sections will be merged. For example, when the memory used by p1 is freed with

```
D p1
```

then that section will merge with the two free sections surrounding it:



Input file format:
The first line will contain the amount of free memory M. The next line will contain the number of allocation/deallocation requests R. The next R lines will be the requests themselves. You may assume that every request will be satisfiable.

Example input file:

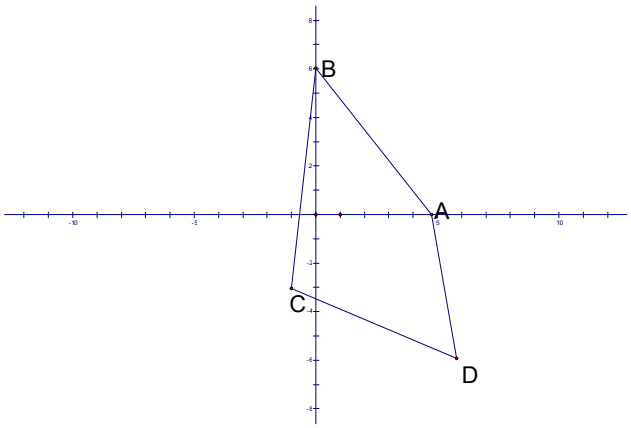
```
100
5
A p0 45
A p1 20
D p0
A p2 10
D p1
```

After each request, your program should reprint the request and then print the contents of memory showing the process name, the beginning memory address of that "chunk", and the ending memory address of that "chunk". The output for the above example would be

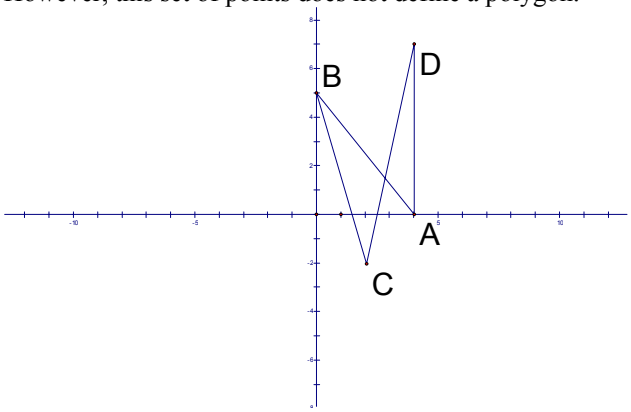
```
A p0 45
p0      0       44
free    45      99

A p1 20
p0      0       44
p1      45      64
free    65      99

D p0
free    0       44
p1      45      64
free    65      99

A p2 10
p2      0       9
free    10      44
p1      45      64
free    65      99

D p1
p2      0       9
free    10      99
```

6. Polygons

Suppose we choose 4 points—A, B, C, and D—and construct the line segments AB, BC, CD, DA. Define the figure formed by these segments to be the points enclosed by the line segments. This figure might form a polygon or might not. We define a polygon to mean it is possible to connect any two points in the polygon by a line or curve without leaving the interior of the polygon. For example, the figure formed by A, B, C, and D below is a polygon.



However, this set of points does not define a polygon.



Your task is to write a program which takes one or more sets of points and, for each set, determines if the points form a polygon. The first number on each line will be the number of points. After this, the points will be given in order. Each point will be represented as its x coordinate and then its y coordinate. All coordinates will be integers. The last line of input will have a 0 on it and merely indicates the end of the input.

Example input file:

```
4 5 0 0 6 -1 -3 6 -6        This is first graph above.
4 4 0 0 5 2 -2 4 7          This is second graph above.
0
```

The output should be:

```
Set 1    Yes
Set 2    No
```