Tenth Annual
University of Central Florida

ACM • UPE

High School Programming Tournament

Problems

Problem Name	Filename
Check-ing Hai-ku	HAIKU
Run for It!	RUNNING
Simple Polygons	SIMPLE
Good Passwords	PASSWORD
Palindrome Compression	COMPRESS
2001	SPACE
Miss You!	CARDS
Full Moon Predictor	PREDICT
Security Clearance	SECURITY

Call your program file: *Filename*.PAS, *Filename*.C, or *Filename*.CPP
Call your input file: *Filename*.IN

For example, if you are solving Miss You!:

Call your program file: CARDS.PAS, CARDS.C, or CARDS.CPP
Call your input file: CARDS.IN

Check-ing Hai-ku

Filename: HAIKU

A common English homework assignment is to write a Haiku. For our purposes, a Haiku is a poem containing exactly three lines and a total of 17 syllables (first line five, second line seven, and third line five syllables). Because of classroom overcrowding, one teacher wants to automate the task of grading student Haiku assignments.

The Problem

Given homework assignments from several students, determine whether each assignment is a valid Haiku. If it is not, explain why not.

The Input

Each student assignment will appear in the file over one or more lines. A single blank line will follow each student assignment. No assignments will exceed 10 lines in length, and each line will contain 1 to 80 characters. Lines will contain upper and lower case letters, spaces, and the hyphen (as explained below) but will not contain any other characters. A single hyphen will be used within words of more than one syllable to delimit syllables. For example, the word "example" will appear in the input as "ex-am-ple".

The Output

For each student assignment, output one of the following:

```
Good Haiku!
Not a Haiku because it has reason.
```

where *reason* is one of

```
the wrong number of lines
the wrong number of syllables on line 1
the wrong number of syllables on line 2
the wrong number of syllables on line 3
```

If an assignment is not a Haiku for more than one reason, output the first applicable reason from the list.

Sample Input

```
The sun shines bright-ly
Glist-en-ing off the cold pool
Of deep blue wat-er

There once was a man from Nan-tuck-et
Who bought from a ven-dor a buck-et
He got a good price
But then he found lice
So much for the luck of the buck-et
```

Sample Output

```
Good Haiku!
Not a Haiku because it has the wrong number of lines.
```

Run for It!

Filename: RUNNING

Five kilometer (5K, or 5000 meters) road races attract runners of all ages. Novice runners start the race at top speed, only to wear out after the first kilometer or so. After a brief rest stop, they start at top speed again and tire soon afterward. They continue this until finishing the race. Experienced runners maintain a steady pace throughout the race without stopping.

The Problem

Given statistics about each runner in a 5K road race, determine the winner and the winning time.

The Input

Data for each race begins with the number of runners. A race with no runners ends the input file. The following statistics will be given about each runner, one value per input line:

```
n, Runner name
s, Running speed (meters per second)
m, Time runner maintains speed (seconds)
r, Rest time (seconds)
```

Runner *n* runs at *s* meters per second for *m* seconds and then must rest for *r* seconds before restarting. The runner's name will not contain a space. Assume each input line starts in column 1.

The Output

Identify each race by its ordering in the input file as shown below:

```
Race race_num: The winner is runner_name with a time of mm:ss.
```

Sample Input

```
2
David
5.0
480.0
5.0
Steve
3.5
400.0
5.0
2
John
5.5
100.5
10.0
Tim
4.9
640.0
5.0
0
```

Sample Output

```
Race 1: The winner is David with a time of 16:50.
Race 2: The winner is John with a time of 16:39.
```

Simple Polygons

Filename: SIMPLE

A polygon is *simple* if none of its edges cross. For example,

Simple Polygon

Not a Simple Polygon

The Problem

Given a closed polygon, determine whether it is simple.

The Input

Each data set begins with a line containing a single integer n ($0 \le n \le 50$; $n \ne 1$ and $n \ne 2$); there are n vertices in the polygon. The next n lines each contain one vertex (x,y) of the polygon, where x and y are integers such that $-100 \le x \le 100$ and $-100 \le y \le 100$. The list of vertices is ordered around the polygon (in other words, the first and second vertices are endpoints of a polygon edge, as are the second and third vertices, etc.). The first and last vertices listed also are endpoints of a polygon edge. Your program must stop processing input data when it reaches a polygon with no vertices.

The Output

For each input polygon, output a single line containing either SIMPLE or NOT SIMPLE, whichever is appropriate.

Sample Input

```
3
0 0
10 0
0 10
4
0 0
10 0
10 10
0 -10
0
```

Sample Output

```
SIMPLE
NOT SIMPLE
```

Good Passwords

Filename: PASSWORD

Bonehead told his girlfriend, Airhead, "It's really cool we are using each other's names as our passwords."

Airhead replied, "But some people say you shouldn't use a password that people can guess."

Bonehead said, "If we can't guess it, how are we going to log in?"

Airhead thought about this for a while. "I don't know, I'm getting a headache!"

The Problem

Define a good password as one that:

- is at least five characters long,
  is not all lower-case letters,
- is not all upper-case letters,
- is not all digits, and
- is not an English word.

For simplicity, define an English word as one containing only letters such that every consonant is followed immediately by a vowel (a, e, i, o, u). If the last letter in the word is a consonant, it does not need to have a vowel after it. For example, "good" and "five", are English words, but "password" is not (for example, because no vowel follows the first 's').

Given a string, determine whether or not it is a good password.

The Input

Each input line contains a string of at least one and at most 50 characters. Assume each string starts in column 1 and does not contain any blanks or other non-printable characters. End of data is indicated by end-of-file.

The Output

Print each input string. On the next line, print a message indicating whether or not the input is a good password. Leave a blank line after the output for each set. Follow the format illustrated in the Sample Output.

See the following page for sample input and sample output for this problem.

Sample Input

```
ali100
OROOJI200
abcdef
ABCDEF
AbCdEf
aEioUae
123456
abc
ToBeToBe
```

Sample Output

```
ali100
Good

OROOJI200
Good

abcdef
Not Good

ABCDEF
Not Good

AbCdEf
Good

aEioUae
Not Good

123456
Not Good

abc
Not Good

ToBeToBe
Not Good
```

Palindrome Compression

Filename: COMPRESS

As the newest member of the development team for the WordQuiteGood word processor, you have been assigned the unpleasant task of optimizing the spelling dictionary. You've decided to save a kilobyte or two of disk space by compressing all the palindromes in the dictionary. A palindrome is a word that is the same when its letters are reversed. Thus, "radar" and "redder" are palindromes. Note an important difference between these two palindromes: the first has an odd number of letters, while the second has an even number.

The Problem

Given a word, "compress" it if it is a palindrome by removing the redundant letters from the end of the word. If the palindrome has an even number of letters, then the entire last half of the word can be removed. If the palindrome has an odd number of letters, the center character is not removed. An extra character must be added to the end of each palindrome so it can be "decompressed" later: an asterisk (*) is appended to the compressed versions of palindromes with an even number of letters, and a tilde (~) is appended to the compressed versions of palindromes with an odd number of letters.

The Input

Each line of input contains one word, starting in the first input column. There will be no spaces after the word. The word will consist of only uppercase letters (A through Z). Each word will be from 1 to 70 characters in length. End of input will be indicated by end of file.

The Output

Output a "compressed" version of each input palindrome, as described above. If the input word is not a palindrome, output the word exactly as it appears in the input.

Sample Input

```
RADAR
REDDER
BANJO
NOON
EYE
```

Sample Output

```
RAD~
RED*
BANJO
NO*
EY~
```

2001

Filename: SPACE

As you probably know, "HAL" (the computer from *2001: A Space Odyssey*) can be formed from "IBM" by shifting each letter backwards one position in the alphabet (I→H, B→A, and M→L). What other coincidences might we find by shifting letters backward?

The Problem

Given a three letter word containing only uppercase letters, output a new three letter word (also in all uppercase) such that each new letter is one "less" than in the original input. Note that we define Z as being one less than A.

The Input

The first input line contains a positive integer, *n*. There will be *n* data sets in the file, one data set per line. Each data set contains one three letter word starting in the first column.

The Output

For each three letter word, output the new three letter word as described above.

Sample Input

```
3
IBM
UCF
FLA
```

Sample Output

```
HAL
TBE
EKZ
```

Miss You!

Filename: CARDS

The CIA (Central Imperfection Agency), using our good tax money, is responsible for making sure decks of playing cards are complete before sending them to the FBI (Federal Boring Investigators). Considering how inept these agencies are, you write a program to help them.

The Problem

Given a deck of cards, your program is to determine which cards are missing from the deck.

The Input

There will be multiple data sets in the input. Each data set starts on a new input line and will be given on one or more lines. Input lines will not exceed column 70, and each line will contain some data. Each card is represented by two adjacent characters, i.e., there won't be spaces between the two characters. The cards on an input line are separated from each other by at least one space. The first character of a card will be 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, or A, representing two through ace, respectively. The second character will convey the suit using initial letter of S, H, D, or C for spades, hearts, diamonds, and clubs. End of a data set is indicated by **. End of data is indicated by end-of-file.

The Output

Print a heading for each data set. Then print the missing cards (in order) on four output lines. Leave a blank line after the output for each data set. Follow the format illustrated in the Sample Output.

Sample Input

```
2S 3S 4S 5S 6S 7S 8S 9S TS JS QS KS AS
2H 3H    5H 6H 7H    9H
2D 3D 4D 5D 6D 7D 8D 9D TD JD QD KD AD
**
TS JS QS KS AS
2S 3S 4S 5S 6S 7S 8S 9S
2H 3H    5H 6H 7H    9H
2D 3D 4D 5D 6D 7D 8D 9D TD JD QD KD AD
2C 3C 4C 5C 6C 7C 8C 9C
TC JC QC KC AC **
```

Sample Output

```
Data set 1:
   Spades:
   Hearts: 4 8 T J Q K A
   Diamonds:
   Clubs: 2 3 4 5 6 7 8 9 T J Q K A

Data set 2:
   Spades:
   Hearts: 4 8 T J Q K A
   Diamonds:
   Clubs:
```

Full Moon Predictor

Filename: PREDICT

Thirty days hath September,
April, June, and November.
All the rest have thirty-one,
Save February which has twenty-eight...

Part of your job as treasurer of the interpretive dance club is to plan the schedule for the year. You have decided to hold the Out Of Doors Nature Dances on every full moon. This will give you the most light possible and offer inspiration to your interpretive dance-club friends. Thus, you need to know what days the moon will be full. Assume for the purposes of this problem that the moon is full every 29 days.

The Problem

Given the date of the first full moon this year, determine the other full moon dates this year. Assume this is not a leap year.

The Input

There will be multiple data sets. Each set contains one integer on its own line. The integer indicates the date of the first full moon, which will be a day in January. Your program must stop processing input when the date is 0.

The Output

For each full moon date after the one given in the input, output the message "`Full Moon On` *`month-day`*", where *`month`* is the integer value of the month (without padding; that is, February is represented as "2", not as "02"), and *`day`* is the integer value of the date (again without padding). Print a blank line after the output for each data set.

Sample Input

```
3
0
```

Sample Output

```
Full Moon On 2-1
Full Moon On 3-2
Full Moon On 3-31
Full Moon On 4-29
Full Moon On 5-28
Full Moon On 6-26
Full Moon On 7-25
Full Moon On 8-23
Full Moon On 9-21
Full Moon On 10-20
Full Moon On 11-18
Full Moon On 12-17
```

Security Clearance

Filename: SECURITY

MicroSock Industries' security system uses automatic human identification. People wanting to enter secure areas are scanned by a computer system to confirm their permission. Scanning may involve retina, fingerprint, voice, or face recognition, or other methods. However, the programs in the current system are not foolproof, and are not always able to distinguish between similar people. In such cases, the software identifies a list of possible employees that might be the person attempting to gain access.

The MicroSock security computer maintains a database of access areas for each employee. If all employees in the possible-list have access to an area, the security system grants access to the area. However, if some employees in the list do not have access to an area, then the system refuses access for the candidate.

The Problem

Given the security access database and the possible-list for a candidate, determine all areas that the security system will allow access for the candidate.

The Input

The input has two parts: the security access database followed by a series of possible-lists. Note that no line of input will be longer than 80 characters.

The first line of the security database is a positive integer n which specifies how many people are in the security database. Following this first line, there are $2n$ lines of text, a pair of lines for each person in the database. The first line of each pair is the name of a person. The second line is a list of zero or more locations that the person is authorized to enter. The locations consist only of alphanumeric characters and are separated by spaces.

The first line of the series of possible-lists is a positive integer m specifying how many possible-lists are to be processed. After this line are m possible-lists. The first line of each possible-list is a non-negative integer k indicating how many different people the scanned individual might be. On the following k lines of input are the possible identities of the scanned person, one per line.

The Output

For each possible-list, print a single line of text indicating which areas the scanned person definitely has access to with the following format:

```
Grant access to area, ..., area.
```

If there are no permissible areas for the person, print the following:

```
Do not grant access to any areas.
```

See the following page for sample input and sample output for this problem.

Sample Input

```
3
Ali Orooji
DataProcessing DatabaseResearch AILab
Mike Smith
VisualProgramming DataProcessing
Travis Terry
VirtualReality SimulationLab ParallelProcessingLab
3
1
Ali Orooji
2
Ali Orooji
Mike Smith
2
Ali Orooji
Travis Terry
```

Sample Output

```
Grant access to DataProcessing, DatabaseResearch, AILab.
Grant access to DataProcessing.
Do not grant access to any areas.
```