2002 APICS Programming Contest

The 2002 Atlantic Canada programming Contest was held at Mount Allison University in Sackville NB Friday Oct 18 2002. There were 16 teams of three students from nine universities in the four Atlantic provinces.

Each team of 3 students was given 6 problems to try to solve in 5 hours using one computer following the rules for the ACM programming contests.

Each problem has sets of associated data files and matching output files. Files 1 and sometimes 2 are available to the contestants during the competition. The others are reserved for the judges to test submitted solutions.

a. My Grammar always said
b. Searching Binary Trees
c. Car Race
d. Circular prerequisites
e. Hypercodes
f. The Lucky Number Sieve

The participating teams came from the following universities:

```
University of New Brunswick, Fredericton Campus.
University of New Brunswick, St John Campus.
Universite de Moncton , Moncton N.B.
Mount Allison University, Sackville N.B.

Acadia University, Wolfville N.S.
St Marys University, Halifax  N.S.
St Francis Xavier University, Antigonish N.S.

University of Prince Edward Island, Charlottetown.

Memorial University of Newfoundland, St. John's.
```

The top 4 teams from this competition advance to the next level of competition in Rochester N.Y.

Problem A: As Grammar Always Said ...

One way to get computers to generate English text (hopefully intelligible) is to use a grammar, which is a set of rules defining how sentences are constructed. Given such a set of rules, together with an input sentence, it is natural to ask the question, "Is this sentence generated by this grammar?" This is the question you are asked to answer in the problem below.

For our purposes, a rule in the grammar will always have one of the two forms:

<variable> -> <word>
<variable> -> <variable> <word>

A <variable> is an uppercase letter (A,B,C,...) and a <word> is a lowercase English word. The only other element is the arrow symbol (->). (Consecutive rule elements are separated by a single white space.) The variable S, called the start symbol, plays a special role. Note that there will always be at least one rule which begins "S ->".

A complete sentence consists entirely of English words. An incomplete sentence contains at least one variable.

To construct a complete sentence:

1. begin with the incomplete sentence consisting of the start symbol S
2. given an incomplete sentence containing some variable, say X, pick any rule that begins "X ->" and replace X in the sentence with whatever appears to the right of the arrow (->) in the rule
3. stop if the sentence is complete, else goto 2 (yes, that really was a "goto")

Input: A list of rules, one per line, followed by one or more sentences, one per line.

Sample input (available in file A1.dat):
```
A -> A walk
B -> C love
C -> D walk
C -> you
A -> C cats
S -> B cats
B -> D talk
A -> C love
S -> A cats
B -> B love
B -> B walk
D -> i
A -> they
you love cats
i walk cats
```

Output: One line for each sentence in the input file. An output line consists of the word "yes" if the corresponding sentence in the input file is generated by the grammar, and "no" otherwise.

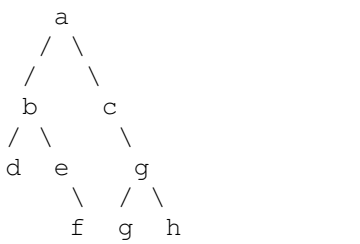
Sample output (A1.out):
```
yes
no
```

There is also a second set of input/output files (A2.dat/A2.out).

Problem B: Binary Trees

Consider the following encoding of a binary tree:
    - a single letter
    - (left subtree) single letter (right subtree)
Note that subtrees may be empty.

For example, the binary tree



is given by the following expression: ((d)b(()e(f)))a(()c((g)g(h)))

Write a program that finds the path from the root to a given node. The output must be a "T" (for Top node) followed by a sequence of "R" (for right) and "L" (for left). You may assume that the node will always be in the tree. The input consists of a single letter followed by the binary tree

For example, with the following input (available as file B1.dat):

```
f
((d)b(()e(f)))a(()c((g)g(h)))
```

the output should be (available in file B1.out)

```
TLRR
```

For the input in file B2.dat:

```
a
((d)b(()e(f)))a(()c((g)g(h)))
```

the output would be (as in file B2.out)

```
T
```

Problem C: Car Race

Consider the following sample input (available as file C1.dat)

```
10 20.       30 40. 40 40.        60 20.        80 10.
      20 20.        40 60. 50 40. 60  0. 70 10. 80  0. 90 20.
```

Each line contains a sequence of time-speed pairs of values. The times are integer and the velocity/speeds have a decimal point. The first line gives the speed of car A at various times. The second line contains similar data for car B. Note that data for the two cars may be at different times.

For times in between given data points the acceration is constant which implies that the speed varies linearly. For example, from time 0 to 10 A's speed increases from 0. to 20. so at an in-between time, such as 6, the speed would be 12. At time 50 we see that A's speed would be 30.

At time 0 the speed is 0.0 for both cars.
We presume that for times after the last data point the speed is constant.

The output from your program for this data should be (file C1.out):

```
B passed A during 46th second
A passed B during 61th second
B passed A during 111th second
```

Your program should calculate each time where one car overtakes the other. Distances which differ by less than eps=0.0000000001 are considered equal. A overtakes B during a one second period if A was more then eps behind and moves to more than eps ahead during that second.

There will not be more than 50 data points given for each car.

At the start both cars are tied and when one moves ahead it is not considered an overtaking.

Files C2.dat and C2.out are also available for testing.

Problem D: Prerequisites

In this problem an input is a set of prerequisite rules such as:

```
CS2 CS1
CS2 Math100 Math150
Math200   Math100 Math150
CS212 Math100 Logic112 Phil112
CS212 CS2
CS212 Engl100 Engl150
```

In general, each line of input consists of two or more tokens separated by one or more blank characters. Before taking the course represented by the first token in a rule at least one of the courses represented by the following tokens must be completed.

The first two lines tell us that before doing CS2 you should do CS1 and either Math 100 or 150. Math200 also requires one of Math 100 or 150. Before doing CS212 a person should complete CS2 and one of, Logic112, Phil112, or Math100, and one of Engl100 or Engl150.

The output should show all the courses grouped into lines. E.g.:

```
CS1 Engl100 Engl150 Logic112 Math100 Math150 Phil112
CS2 Math200
CS212
```

All courses take one term to complete. A course belongs on line n if the earliest it can be taken by choosing prerequisites carefully is term n. The courses on the first line can be taken in the first term as they have no prerequisite listed. The prerequisite rules for courses in line n > 1 will all require at least one of the courses from the previous line. Within a line the courses are listed in lexicographic order.

You may assume that such an ordering is possible, i.e., that there is no pair of courses where each is an unavoidable prerequisite of the other.

The sample input and output above are available for testing in file D1.dat and D1.out. Also available is file D2.dat:

```
F O X
C A T
D O G
A P E
E M U
G N U
A N T
B A T
C A N
F O G
G A F
```

for which the output should be (file D2.out):

```
M N O P T U X
A B C D E F
G
```

Note that C is in the second line rather than the third because its pre-requisites may be satisfied by taking T and N which are both in the first line. Also in the last two rules F and G are prerequisites of each other but there are alternative choices which avoid this circularity.

Problem E: HYPERCODES

A *hypercode* is a set of one or more strings with the following property. For any two different strings S and T in the hypercode, if any number of characters is inserted in S, then the resulting string is different from T. For example, the strings abbab and aabbbaabba cannot be part of a hypercode, as the second string can be obtained from the first one by inserting the characters **a**, **b**, **a**, **b**, and **a** as follows: a**a**bb**b**a**a**b**ba**. On the other hand, the strings abbaaab, aabbbaab, and aaabbbbab constitute a hypercode. Write a program that reads from standard input several sets of strings and outputs on the screen, for each set, yes or no depending on whether the set is a hypercode. Each set of n strings occupies 1+n consecutive lines in the input file as follows:

n
S1
S2
...
Sn

where n is the number of strings in the set and S1, S2, ..., Sn are the n strings. The number n and the strings Si start at the beginning of the lines - the end of line character is not part of any string. The input file ends with the number 0. Here is a typical input/output situation:

>Only one input file(E1.dat) is available for your testing:

2
abbab
aabbbaabba
3
aaabbbbab
abbaaab
aabbbaab
0

The output from this (file E1.out) should be:

no
yes

Although in the above examples strings consist of a's and b's only, your program should be able to handle strings containing any printable character.

Problem F: Sieve

You should be feeling at least a little bit lucky when you tackle this problem, because your job here is to find some "lucky" numbers. A "lucky" number is simply a positive integer from a given set of contiguous positive integers that survives the following process of elimination.

Your program first inputs two values, a low value, and a high value, in that order, that determine the range of values in the contiguous set of positive integers.

Look at the first number in the set. Suppose it's a. Eliminate every a-th value from the set. Now consider the remaining set, and look at the second value in it. Suppose this value is b. Eliminate every b-th value from the current set. Once again, consider the remaining set and now look at the third value in it. Suppose it's c. Eliminate every c-th remaining value. And so on ...

Continue this process until no values can be eliminated. Then print out the values in the remaining set, 10 per line, right justified, in a field of width n spaces where n is one more than the number of digits in the largest number.

Please be careful not to have any spaces at the end of an output line.

Sample Input/Output

F1.dat
```
4 60
```

F1.out
```
  4  5  6  8 10 13 14 18 20 25
 26 28 34 38 40 44 46 53 54 58
 60
```

F2.dat
```
2 100
```

F2.out
```
   2   4   6  10  12  18  20  22  26  34
  36  42  44  50  52  54  58  68  70  76
  84  90  98 100
```