Problem 8: Semigroups (aka 18-Wheeler Caravans)[1]

Source filename:	`semigrp.cpp`
Input filename:	`semigrp.in`
Output filename:	`semigrp.out`

Definitions

A binary operation on a set S is a function that assigns to each ordered pair of elements of S a unique element of S. We often use some special symbol (such as * or +) to represent a binary operation. For example, if we use the symbol '#' to represent some arbitrary binary operation on the set S = {a,b,c}, then a#b equals some element of S (as does b#a, a#a, a#c, and every other possible combination).

From the above definition, it would follow that the normal definitions for addition, subtraction, and multiplication are all binary operations when defined on the set of all integers. However, division is not a binary operation for the set of integers, since $1 \div 2$ yields 0.5 which is not an integer.

The use of the word "ordered" in the definition for binary operations is important, for it allows the possibility that the element assigned to a#b may be different from the one assigned to b#a. In the case of integers, this is evident with the binary operation we know as subtraction, since 5 − 3 is not equal to 3 − 5. If in a particular case, x#y = y#x for all elements x and y in the set, we say that the binary operation is commutative. The standard addition operation on the set of integers is commutative.

For the remainder of this problem we will only concern ourselves with small sets (1 to 26 elements). For small sets such as these, the unique assignments that define an operation can be expressed by simply writing down all possible assignments in a "Cayley multiplication" table. For instance, the binary operation '#' on the set S = {a,b,c} might be defined by:

```
# | a b c
---------
a | b c b
b | a c b
c | c b a
```

The left column of the table represents the 1st number in an ordered pair, and the top row represents the 2nd. Thus, in this example, a#b=c, b#a=a, and c#c=a. Notice that the body of the table consists solely of elements from the set S, which must be true for any binary operation. Also notice that since b#a ≠ a#b, this operation is not commutative,.

A binary operation, #, on a set S is associative if (x#y)#z = x#(y#z) for all elements x, y, and z in the set S. In the example with the table in the previous paragraph, the operation is not associative, since (a#b)#c ≠ a#(b#c). If a binary operation, #, on a set is associative, then we say that the pair <S,#> forms a semigroup. If the binary operation is commutative as well as associative, then we say that the semigroup is commutative.

[1] This problem appeared on the 1996 ACM Mid-Central Regional Programming Contest.

Input / Output File Specifications

Write a program that will read the elements of sets together with corresponding "multiplication" tables which denote possible binary operations. Your program should then determine if the set S with the defined operation constitutes a semigroup. If the set and operation do not form a semigroup, your program should report that the pair does not form a semigroup and state why. If the set and operation pair does form a semigroup, your program should check to see if the semigroup is also a commutative semigroup.

You should note from the first paragraph on the page 1 that a binary operation should assign an element from S to each pair of elements of S. That is, if x and y are elements in the set S, then x#y should also be an element in the set S. We say that the operation is "closed". S is NOT a semigroup if this property is not true for a given multiplication table. If you discover a value in the body of the multiplication table that isn't an element of the set S, then you program should report a message similar to:
`NOT A SEMIGROUP: x#y = z WHICH IS NOT AN ELEMENT OF THE SET`
If more than one such counter-example exists, you may simply use any one of them.

If the multiplication table does correspond to a closed binary operation, then your program should check to see if it satisfies the associative property. If not, then your program should report a message similar to:
`NOT A SEMIGROUP: (x#y)#z IS NOT EQUAL TO x#(y#z)`
If more than one such counter-example exists, you may simply use any one of them.

Note that a counter-example of the closed property takes precedence over any counter-examples of the associative property.

If the multiplication table corresponds to a closed, binary operation that has the associative property, then the set together with the operation constitutes a semigroup. Your program should then check to see if the operation is also commutative.

If it isn't commutative, then the program should report a message similar to:
`SEMIGROUP BUT NOT COMMUTATIVE (x#y IS NOT EQUAL TO y#x)`
If more than one such counter-example exists, you may simply use any one of them.

And if, on the other hand, the set and operation correspond to a commutative semigroup, then the program should simply report:
`COMMUTATIVE SEMIGROUP`

Thus, for each test case one of the following four results should be reported:
```
NOT A SEMIGROUP: x#y = z WHICH IS NOT AN ELEMENT OF THE SET
NOT A SEMIGROUP: (x#y)#z IS NOT EQUAL TO x#(y#z)
SEMIGROUP BUT NOT COMMUTATIVE (x#y IS NOT EQUAL TO y#x)
COMMUTATIVE SEMIGROUP
```

The first line of the input file contains a single integer, *n* where ($1 \leq n \leq 26$).

The next line of the input file will contain *n* unique, lower case letters of the alphabet. These letters represent the elements of the set. Although each letter is unique (no duplicates), they are not necessarily arranged in alphabetical order.

The next *n* lines contain the body of the "multiplication" table that corresponds to the elements in the previous line. Each of these lines will contain *n* lower case letters. For example, the 1st such line corresponds to the first row of the body of the table. We will assume that the ordering of the rows and columns of the table coincide with the ordering in the line that defines the elements of the set.

After the table, the input file will contain a line with a single integer, *n* where ($0 \leq n \leq 26$). If $n > 0$, then this *n* corresponds to another set and corresponding table contained in the next *n*+1 lines that should be reported. If $n = 0$, then you have reached the end of the input file.

The output file should contain the following for each set and table found in the input file:

1. List of the elements of S in same order as found in the input file using the following format: `S = {a,b,c,d}`
2. A line that starts with the characters ‘ `#|` ’ (notice the leading space) followed by the *n* elements of the set (no spaces or commas). For example: `#|abcd`
3. A line that begins with the characters ‘ `-+` ’ (again, notice the leading space) followed by *n* more dashes ‘`-`’. For example: `-+----`
4. List of the *n* rows and columns of the “multiplication” table in the same order as found in the input file. The i^{th} line of the table should begin with a space followed by the i^{th} element of the set followed by the ‘ `|` ’ character followed by the *n* characters in the i^{th} row of the body of the table (no spaces). For example: `a|abcd`
5. One blank line.
6. One line that reports what your program found to be true. This must be one of the four possible results listed above. Be sure to substitute the actual elements of the set that yield any counter-examples.
7. A line of 30 dashes.
8. One blank line to separate this report from subsequent reports.

Sample Input File

```
3
abc
abc
bca
cab
3
abc
abc
bca
cad
4
acdb
aaaa
aaca
aada
aaab
5
abcde
aaaaa
bbabb
cccbc
ddddd
eeeee
0
```

Sample Output File

```
S = {a,b,c}
 #|abc
 -+---
 a|abc
 b|bca
 c|cab

COMMUTATIVE SEMIGROUP
------------------------------

S = {a,b,c}
 #|abc
 -+---
 a|abc
 b|bca
 c|cad

NOT A SEMIGROUP: c#c = d WHICH IS NOT AN ELEMENT OF THE SET
------------------------------

S = {a,c,d,b}
 #|acdb
 -+----
 a|aaaa
 c|aaca
 d|aada
 b|aaab

SEMIGROUP BUT NOT COMMUTATIVE (c#d IS NOT EQUAL TO d#c)
------------------------------

S = {a,b,c,d,e}
 #|abcde
 -+-----
 a|aaaaa
 b|bbabb
 c|cccbc
 d|ddddd
 e|eeeee

NOT A SEMIGROUP: (b#a)#c IS NOT EQUAL TO b#(a#c)
------------------------------
```