ODU-ACM Spring 1992 Programming Contest

Old Dominion University
Department of Computer Science
Norfolk, Virginia 23529-0162

February 15, 1992

1. Linear Cellular Automata

A biologist is experimenting with DNA modification of bacterial colonies being grown in a linear array of culture dishes. By changing the DNA, he is able "program" the bacteria to respond to the population density of the neighboring dishes. Population is measured on a four point scale (from 0 to 3). The DNA information is represented as an array `DNA`, indexed from 0 to 9, of population density values and is interpreted as follows:

- In any given culture dish, let K be the sum of that culture dish's density and the densities of the dish immediately to the left and the dish immediately to the right. Then, by the next day, that dish will have a population density of `DNA[K]`.
- The dish at the far left of the line is considered to have a left neighbor with population density 0.
- The dish at the far right of the line is considered to have a right neighbor with population density 0.

Now, clearly, some DNA programs cause all the bacteria to die off (e.g., [0,0,0,0,0,0,0,0,0,0]). Others result in immediate population explosions (e.g., [3,3,3,3,3,3,3,3,3,3]). The biologist is interested in how some of the less obvious intermediate DNA programs might behave.

Write a program to simulate the culture growth in a line of 40 dishes, assuming that dish 20 starts with a population density of 1 and all other dishes start with a population density of 0.

Input

Your program will read in the DNA program (10 integer values) on one line.

Output

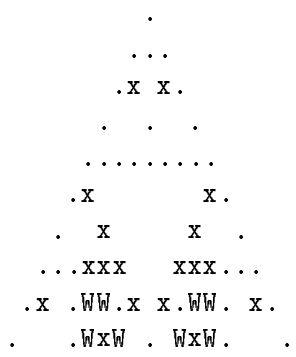
It should print the densities of the 40 dishes for each of the next 50 days. Each day's printout should occupy one line of 40 characters. Each dish is represented by a single character on that line. Zero population densities are to be printed as the character ' '. Population density 1 will be printed as the character '.'. Population density 2 will be printed as the character 'x'. Population density 3 will be printed as the character 'W'.

Example

Given the input:

```
0 1 2 0 1 3 3 2 3 0
```

The first ten lines of output should be



2. The Decoder

Write a complete program that will correctly decode a set of characters into a valid message. Your program should read a given file of a simple coded set of characters and print the exact message that the characters contain. The code key for this simple coding is a one for one character substitution based upon a single arithmetic manipulation of the printable portion of the ascii character set.

For example: with the input file that contains:

```
1JKJ'pz'{ol'{yhklthyr'vm'{ol'Jvu{yvs'Kh{h'Jvywvyh{pvu5
1PIT'pz'h'{yhklthyr'vm'{ol'Pu{lyuh{pvuhs'I|zpulzz'Thjopul'Jvywvyh{pvu5
1KLJ'pz'{ol'{yhklthyr'vm'{ol'Kpnp{hs'Lx|pwtlu{'Jvywvyh{pvu5
```

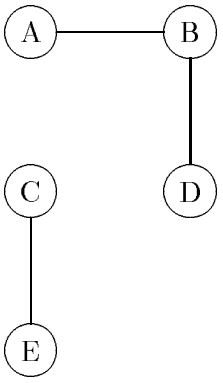
your program should print the message:

```
*CDC is the trademark of the Control Data Corporation.
*IBM is a trademark of the International Business Machine Corporation.
*DEC is the trademark of the Digital Equipment Corporation.
```

Your program should accept all sets of characters that use the same encoding scheme and should print the actual message of each set of characters.

3. Graph Connectivity

Consider a graph G formed from a large number of nodes connected by edges. G is said to be *connected* if a path can be found in 0 or more steps between any pair of nodes in G. For example, the graph below is not connected because there is no path from A to C.



This graph contains, however, a number of subgraphs that are connected, one for each of the following sets of nodes: $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$, $\{A, B\}$, $\{B, D\}$, $\{C, E\}$, $\{A, B, D\}$

A connected subgraph is *maximal* if there are no nodes and edges in the original graph that could be added to the subgraph and still leave it connected. There are two maximal connected subgraphs above, one associated with the nodes $\{A, B, D\}$ and the other with the nodes $\{C, E\}$.

Write a program to determine the number of maximal connected subgraphs of a given graph.

Input:

The first line of input contains a single uppercase alphabetic character. This character represents the largest node name in the graph. Each successive line contains a pair of uppercase alphabetic characters denoting an edge in the graph. Input is terminated by a blank line.

Output:

Write the number of maximal connected subgraphs.

EXAMPLE:

A possible input set for the graph pictured above is:

```
E
AB
C E
DB
E  C
```

The output is 2.

4. Overlapping Rectangles

When displaying a collection of rectangular windows on a SUN screen, a critical step is determining whether two windows overlap, and, if so, where on the screen the overlapping region lies.

Write a program to perform this function. Your program will accept as input the coordinates of two rectangular windows. If the windows do not overlap, your program should produce a message to that effect. If they do overlap, you should compute the coordinates of the overlapping region (which must itself be a rectangle).

All coordinates are expressed in "pixel numbers", integer values ranging from 0 to 9999. A rectangle will be described by two pairs of (X,Y) coordinates. The first pair gives the coordinates of the lower left-hand corner (XLL,YLL). The second pair gives the coordinates of the upper right-hand coordinates (XUR, YUR). You are guaranteed that XLL $<$ XUR and YLL $<$ YUR.

Input

Two lines. The first contains the integer numbers XLL, YLL, XUR and YUR for the first window. The second contains the same numbers for the second window.

Output

If the two windows do not overlap, print the message "No Overlap". If the two windows do overlap, print 4 integer numbers giving the XLL, YLL, XUR and YUR for the region of overlap.

Note that two windows that share a common edge but have no other points in common are considered to have "No Overlap".

Example

Given the input

```
0 20 100 120
80 0 500 60
```

The output should be

```
80 20 100 60
```

5. The Reservation Maker

Write a program that will assist the receptionist in seating customer parties at tables and booths in a large restaurant. Your program should accept for the receptionist the arrival time, the size of the party, whether the party desires a table or a booth, in the smoking or nonsmoking sections of the restaurant. For each request in a list of requests, the program should provide a table number and the approximate waiting time before the party can be seated. The restaurant has the following seating characteristics:

Nonsmoking			**Smoking**		
Tables	Kind	Capacity	Tables	Kind	Capacity
1–4:	booths	6	5–10:	booths	6
11–20:	tables	4	21–30:	tables	4
31–40:	tables	4	41–50:	tables	4
51–55:	tables	6	56–60:	tables	6
61–65:	tables	2	66–70:	tables	2

Each table of the table groups is portable and can be moved such that as many as five tables of any group may be connected to seat a larger party than any one table could seat. This connection of tables is only possible if the sequential number of the tables to be used are vacant. Thus 2 four-person tables can be joined together to seat a party of 6; 3 four-person table can be joined to seat a party of 8, and so on.

Parties of 1 take 35 minutes, parties of 2 take 47 minutes at the restaurant (service is very predictable), parties of 3 or 4 take 52 minutes, while parties of 5 to 10 take 55 minutes. Parties of greater than 10 take (size of party $* 5 + 16$) minutes.

Your program should always assign the lowest booth/table number available. You should also process the requests in the order given and once an assignment is made, not change it.

Input

Your program should accept a sequence of reservation requests (the restaurant requires reservations). Each reservation consists of one line with an integer arrival time (in minutes past 8:00 p.m.), the party size (at least 1), a blank, then an 'S' or an 'N' (smoking or nonsmoking), a blank, then a 'B' or a 'T' (booth or table).

Output

For each reservation request which can be satisfied, your program should then print the list of table numbers or the table/booth number and the number of minutes the party will have to wait after their arrival. If a request cannot be satisfied, print 'Impossible.'

Example

For the input:

```
60 5 N B
0 4 N B
10 3 N B
10 2 N B
13 2 N B
0 6 N B
30 12 N T
60 8 S B
```

the output would be:

```
1 0
2 0
3 0
4 0
1 0
2 47
11 12 13 14 15 0
Impossible
```