The Great Computer Challenge
Pascal
Level IV

Directions: Write programs to implement the specifications on the following pages. All programs count equally. By far the greatest factor in evaluating your programs is whether they work correctly. Additional credit may be earned by good style and documentation, pleasant user interface, and clever approaches to problems.

The Great Computer Challenge
Pascal
Level IV

Problem 1: Where is MY ROBOT?

The problem is one of find the coordinates of the the missing Robot. The Robot is known to have started at the Cartesian coordinates of (0,0) and to have made a series of moves as dictated by his input sequence. Each step in the input sequence is one of the words east***, north, south*** or ***west***. For an input of east the Robot moves one unit in the positive X direction and of course to go west the Robot moves one unit in the negative X direction. Similarly a movement to the north causes the X value to increase by one while south causes the Y value to decrease by one.

Given an input sequence, your program must tell us the final coordinates of the Robot.

Input to your program will be a series of directions, ***east, north, south*** or ***west*** separated by blanks.

The output from your program will be the final location of the Robot in Cartesian coordinates.

Sample
Input
west south east east east north north

Output
(2,1)

Illustration of the sample

```
                             (2,1)
                                ↑
                              | h
                              | t
                              | r
          tsew                | o
(-1,0)<----- (0,0)            | n
s  |                          ↑
o  |                          | h
u  |                          | t
t  |                          | r
h  |                          | o
   ↓     east  east   east    | n
 (-1,-1)-----> -----> ----->(2,1)
```

The Great Computer Challenge
Pascal
Level IV

Problem 2: Perfect Numbers

The Greeks began an examination of numerology by classifying all positive integers as either ***perfect, abundant,*** or ***deficient***. This classification scheme is based on the factors (even divisors) of the number. If the sum if all of the factors of a number (excluding the number itself) equals the number then it is said to be ***perfect***. For example, the factors of 6 are 1, 2, 3, and 6. Therefore, the number 6 is a perfect number. The total of the factors of 6 (excluding 6 itself) is 1 + 2 + 3 = 6.

An ***abundant*** number is one in which this sum of factors (excluding the number itself) is greater that the number. An example of an abundant number is 12 because 1 + 2 + 3 + 4 + 6 = 16 > 12. All numbers that are neither perfect nor abundant are ***deficient***.

This program accepts a sequence of integers, classifies each as abundant, perfect, or deficient and displays the factors of the number. It terminates when given an input of zero.

INPUT

Each input value is prompted by the line: ***Specify a positive integer***

Prompted input consists of single integer. An input of zero (0) signals termination. You may assume that all input is correct. Any incorrect input produces undefined results.

OUTPUT

Following the input a line like "***This number is*** < *type* > ***(factors given below)***." is output, where < *type* > is either ***perfect, abundant,*** or ***deficient***. as is appropriate for the input. On the following line, the factors of the input integer are shown.

EXAMPLE

Specify a positive integer: 6
This number is perfect (factors given below).
1 2 3
Specify a positive integer: 12
This number is abundant (factors given below).
1 2 3 4 6
Specify a positive integer: 333
This number is deficient (factors given below).
1 3 9 37 111
Specify a positive integer: 0

The Great Computer Challenge
Pascal
Level IV

Problem 3: Text Compression

Most text files contain many repeated patterns. Storing these files can take up lots of space. You will write a text file compression program that will do two kinds of compression: replace triads (three specified characters in a row) with special characters and replace strings of 5 or more occurrences of any single character with a special code.

Your program will read in 5 triads to be used in compression. Each will be three lower case letters with no intervening blanks. Every time you encounter one of these triads in the input text, you will replace it by an upper case letter corresponding to the place in the input list of triads.

Thus, if the first three triads input to your program were ‘the', ‘and', and ‘ing', then the sentence ‘the dog and the cat were chasing each other' would be compressed to `A dog B A cat were chasC each oAr'.

In addition, if you should find any 5 or more of the same character in a row in the text, you should replace them with a capital Z, followed by the character, followed by the number of occurrences of the character, followed by another capital Z. Thus, the sentence ‘decimal 128 is 10000000 in binary' would be replaced by `decimal 128 is 1Z07Z in binary'. Leading and embedded blanks are to be compressed in the same way if there are 5 or more in a row.

In the case that both types of reduction are possible, you should always perform the triad reduction first. Therefore, using the above table of triads, `theeeee' would be replaced by `Aeeee' and never `thZe5Z'. However, `theeeeee' would be reduced initially to ‘Aeeeee’ and then, because there are 5 e’s, to ‘AZe5Z'.

You should not compress patterns which cross line boundaries.

Input:

The input triads (the first 5 lines of the input file) will start in column 1 and contain three lower case letters per line. Following will be an unspecified number of lines of text for compression. The text will contain only lower case letters, digits, and punctuation. Each line will contain no more than 70 characters.

Output:
The output of your program should be line by line identical to the input text except for the compression described above.

The Great Computer Challenge
Pascal
Level IV

Problem 4: Text Formatting.

The problem is to reformat a paragraph to a specified width. This means that no line in the paragraph should be longer than the width specified and each line should be as long as possible without going over the specified width. In the paragraph, words are separated by blanks and ends of line. No word should be split between two lines in the output. paragraph.

Input

Any number of lines of text, followed by a line which begins with an asterisk (*), followed by a blank, followed by an integer. The integer specifies the width of the paragraph. The asterisk is not considered part of the paragraph.

Output

The input lines of text reformatted to the width specified.

Example

Input

```
This function treats a string
(the value of EXPR) as a vector of unsigned integers, and
returns the value of the element specified by OFFSET and
BITS.
The function may also be assigned to, which causes
the element to be modified.
* 40
```

Output

```
This function treats a string (the value
of EXPR) as a vector of unsigned
integers, and returns the value of the
element specified by OFFSET and BITS.
The function may also be assigned to,
which causes the element to be modified.
```