

# ND Programming Contest, Fall 2008

ND ACM Computer Club

Sunday, September 21, 2008

# #1: Factor Fiction

## The Problem

Let's begin with a game of Fact or Fiction. Specifically, let's determine whether a particular statement about a number's factors is a fact(or) fiction. A perfect number is a number  $N$  such that the sum of all integers  $n$ ,  $1 \leq n < N$ , that divide evenly into  $N$ , is equal to  $N$  itself. For example,  $6 = 1 + 2 + 3$  and  $28 = 1 + 2 + 7 + 14$  are perfect numbers, but 12 is not because  $1 + 2 + 3 + 4 + 6 = 16$ . You will write a program to test for perfect numbers.

Your program should read one integer from the input,  $M$ , which indicates the number of integers to follow. Then read  $M$  integers from standard input, one per line, and for each, print **FACT** or **FICTION** to standard output on its own line: **FACT** if the integer is a perfect number, **FICTION** if not. Each integer will be greater than zero and less than  $2^{31}$ .

## Sample Input

```
5      // five integers follow
6
28
12
496
1000
```

## Sample Output

```
FACT
FACT
FICTION
FACT
FICTION
```

# #2: Pete's Pizza Party

## The Problem

Pete has a bunch of pieces of pizza and wants to know how many guests he can invite to his party and still fairly divide the pizza. For example, if he has 15 pieces, he could invite two guests, because then each person (including Pete) would get five pieces. Pete's real pizza inventory is, fortunately, much larger than 15 pieces, and he needs a computer's help.

The input for this problem will be a sequence of numbers with the final one being zero. For each nonzero number return a list of the numbers of guests who can come to the party and have every person (including Pete) receive the same number of pieces of pizza. You may assume that the given integers will be less than  $2^{31}$ .

## Sample Input

```
3
15
29
49
23757
0
```

## Sample Output

```
0 2
0 2 4 14
0 28
0 6 48
0 2 7918 23756
```

# #3: Diabolical Diction

## The Problem

A local group of eccentric lexicophiles<sup>1</sup> has come together recently to produce a dictionary of ominous-sounding words that they are calling *Diabolical Diction*. They want to sort their dictionary entries by the ominousness of a word. A word's ominousness has been experimentally determined to be directly proportional to the number of adjacent repeated-letter pairs it has. For example, "doom" has an ominousness of 1, "balloon" has an ominousness of 2 (impending danger, very scary!), but "peace" has an ominousness of 0. In a string of more than two consecutive letters, every consecutive pair counts, so that, for example, "eeeeek" has ominousness of 3.

You are to write a program that sorts a list of words by ominousness, highest first. Read an integer from standard input indicating the number of test cases. Then, for each test case, read an integer indicating the number of words. Read in one word per line. Sort by ominousness, highest first. Any ties in ominousness sorting order shall be resolved by an alphabetic sort order, A before Z. For each test case, write the word list to standard output, one per line.

You may assume no more than 4096 words per test case, and no more than 256 characters per word.

## Sample Input

```
2 // two test cases follow
5 // first test case: five words
doom // score: 1
balloon // score: 2
peace // score: 0
eeeeek // score: 3
arrrrrgh // score: 4
3 // second test case: three words
ordinary // score: 0
boring // score: 0
ominousness // score: 1
```

## Sample Output

```
arrrrrgh
eeeeek
balloon
doom
peace
ominousness
boring
ordinary
```

---

<sup>1</sup>*lexicophile*, n.: a lover of dictionaries. (Alright, so I made that one up. I like it, though.)

# #4: There is no 'E' in Count

## The Problem

E is the most common letter in written English. All the more incredible it is, then, when a sentence (perhaps even a book) can be written without the use of a single E.

Interestingly, quite a few numbers, in written-English form, have no E. For example, 34 – “thirty four” – has no E. Nor does 2034 – “two thousand thirty four.” Your task is to find the  $N$ th such number, where  $N = 1$  corresponds to 2 (“two”),  $N = 2$  gives 4 (“four”), etc.<sup>2</sup> For reference, the powers-of-thousand that you will need to know are thousand, million, billion, trillion, quadrillion, quintillion, sexillion, septillion, octillion.

You may assume that  $1 \leq N \leq 2^{31}$ . You will read a number  $C$  from standard input, indicating the number of test cases. For each test case, read a value for  $N$  from standard input, and print the  $N$ th E-less number to standard output in decimal form.

## Sample Input

```
3 // three test cases
1
2
2000
```

## Sample Output

```
2
4
32000000
```

---

<sup>2</sup>Credit to ACM East-Central North America regionals 2007 for this problem.

# #5: Logically Speaking

## The Problem

A professor of logic at Western Central East State Community College (WCESCC) next door has enlisted your help. He makes logical statements to his introductory legal classes, but none of the lawyers-to-be can determine the truth of the statements. It will take an engineer well-versed in digital logic design to solve this problem. That means you.

The statements come in *sum-of-products form*, which looks like  $ABC + A!B + C$  – which means, “A and B and C, or A and not B, or C.” In general, a “product” is a logical AND conjunction, and the “sum” is a logical OR. The exclamation mark prefixes the following variable with a “not.” All variables are a single letter, and have boolean value (true or false).

Your task is to determine whether a given statement is true for *all* values of the variables. Such a statement is called a “tautology.” The easiest way to do this is to build a truth table, which looks like the following for the above statement:

A	B	C	$ABC + A!B + C$
F	F	F	F
F	F	T	T
F	T	F	F
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	T

Clearly, this is not a tautology. However,  $A+!A$  is:

A	$A+!A$
F	T
T	T

Your program will read an integer, giving the number of test cases. For each test case, read one statement. A statement will consist of an integer, the number of variables (which will always start with “A”), another integer, the number of terms, and then each term as written above. There are no “+” operators between the terms. For each statement, output “T” if it is a tautology or “F” if not.

## Sample Input

```
2 // two test cases
3 3 ABC A!B C // 3 variables, 3 terms, ABC + A!B + C
2 4 AB A!B !AB !A!B // 2 vars, 4 terms, AB + A!B + !AB + !A!B
```

## Sample Output

```
F
T
```

# #6: Breaking the Code

## The Problem

A spy has infiltrated an enemy camp and has found the method your enemies are using to encrypt their messages. You also have 5 encrypted messages which you intercepted when your enemy sent a telegraph.

Your program should read a number  $N$  from standard input, indicating the number of test cases. Then, for each test case, read a series of twenty-six letters, where a plaintext A encrypts to the first letter, B encrypts to the second, etc, up to Z. The next five lines will be strings of characters which the enemy has encrypted, terminated by a period. Your program should decrypt each message and print it to standard output with the terminating period.

## Sample Input

```
1
BCDEFGHIJKLMNOPQRSTUVWXYZA
HPPEMVDL.
UIFDBLFTJBMJF.
BUUBDLBUEBXO.
OFWFHSHPOOBHJWFZPVVQ.
DPOHSBUTVSBMFFUIBYP.
```

## Sample Output

```
GOODLUCK.
THECAKEISALIE.
ATTACKATDAWN.
NEVERGONNAGIVEYOUUP.
CONGRATSURALEETHAXOR.
```