L3/M1, 2005–2006

Algorithmique effective – TD 10
Arithmétique élémentaire

1 Isprime

Exercice 1

1. Écrire un programme calculant le symbole de Jacobi d'un couple (entier, entier impair) : il s'agit de la généralisation vue en cours du symbole de Legendre. Les entrées auront le format suivant : chaque ligne du fichier contient les deux entiers $0 < n, m < 2^{63}$ premiers entre eux, avec m impair. La dernière ligne du fichier est 0 0. La sortie doit être au format suivant : sur chaque ligne, écrire la valeur du symbole de Jacobi des deux entiers.

   Sample Input

   ```
   8 5
   238 3847989
   17 19
   0 0
   ```

   Sample Output

   ```
   -1
   -1
   1
   ```

   Vous testerez votre programme sur le fichier

   `http://perso.ens-lyon.fr/sylvain.perifel/algo_effective/td10/jacobi.in`

   la sortie attendue est dans le fichier `jacobi.out`.

2. Écrire un programme testant si un entier est premier. Les entrées auront le format suivant : chaque ligne du fichier contient un entier $0 < n < 2^{62}$. La dernière ligne du fichier est 0. La sortie doit être au format suivant : sur chaque ligne, écrire `Premier` si le nombre est premier, `Non premier` sinon.
   Sample Input

   ```
   187
   32497
   5
   21
   0
   ```

   Sample Output

   ```
   Non premier
   Premier
   Premier
   Non premier
   ```

   Vous testerez votre programme sur le fichier

   `http://perso.ens-lyon.fr/sylvain.perifel/algo_effective/td10/premiers.in`

   la sortie attendue est dans le fichier `premiers.out`. Une des difficultés est l'exponentiation modulaire en restant dans 64 bits : on pourra d'abord programmer la multiplication modulaire pour cela.

2 Nombres de Carmichael

Exercice 2 Carmichael Numbers. UVa : 10006.
An important topic nowadays in computer science is cryptography. Some people even think that cryptography is the only important field in computer science, and that life would not matter at all without

cryptography. Alvaro is one of such persons, and is designing a set of cryptographic procedures for cooking paella. Some of the cryptographic algorithms he is implementing make use of big prime numbers. However, checking if a big number is prime is not so easy. An exhaustive approach can require the division of the number by all the prime numbers smaller than or equal to its square root. For big numbers, the amount of time and storage needed for such operations would certainly ruin the paella.

However, some probabilistic tests exist that offer high confidence at low cost. One of them is the Fermat test.

Let a be a random number between 2 and $n-1$ (n being the number whose primality we are testing). Then, n is probably prime if the following equation holds:

$$a^n \equiv a\ [n].$$

If a number passes the Fermat test several times then it seems prime with a high probability.

Unfortunately, there is some bad news. Some numbers that are not prime still pass the Fermat test with every number smaller than themselves. These numbers are called Carmichael numbers.

In this problem you are asked to write a program to test if a given number is a Carmichael number. Hopefully, the teams that fulfill the task will one day be able to taste a delicious portion of encrypted paella. As a side note, we need to mention that, according to Alvaro, the main advantage of encrypted paella over conventional paella is that nobody but you knows what you are eating.

Input

The input will consist of a series of lines, each containing a small positive number n ($2 < n < 65000$). A number $n = 0$ will mark the end of the input, and must not be processed.

Output

For each number in the input, you have to print if it is a Carmichael number or not, as shown in the sample output.

Sample Input

```
1729
17
561
1109
431
0
```

Sample Output

```
The number 1729 is a Carmichael number.
17 is normal.
The number 561 is a Carmichael number.
1109 is normal.
431 is normal.
```

3 Oncle Petros et son ordinateur

Exercice 3 Summation of Four Primes. UVa : 10168. Time Limit : 4 seconds.
Euclid proved in one of his classic theorems that prime numbers are infinite in number. But can every number be expressed as a summation of four positive primes ? I don't know the answer. Maybe you can help ! ! ! I want your solution to be very efficient as I have a 386 machine at home. But the time limit specified above is for a Pentium III 800 machine. The definition of prime number for this problem is "A prime number is a positive number which has exactly two distinct integer factors". As for example 37 is prime as it has exactly two distinct integer factors 37 and 1.

Input

The input contains one integer number N ($N \leq 10000000$) in every line. This is the number you will have to express as a summation of four primes. Input is terminated by end of file.

Output

For each line of input there is one line of output, which contains four prime numbers according to the given condition. If the number cannot be expressed as a summation of four prime numbers print the line `Impossible.` in a single line. There can be multiple solutions. Any good solution will be accepted.

Sample Input

```
24
36
46
```

Sample Output

```
3 11 3 7
3 7 13 13
11 11 17 7
```

4 Dessert

DM pour le mardi 9 mai 2006...

4.1 Encore un peu d'arithmétique

Exercice 4 Humble numbers. UVa : 443.
A number whose only prime factors are 2, 3, 5 or 7 is called a humble number. The sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 24, 25, 27, ... shows the first 20 humble numbers.

Write a program to find and print the n-th element in this sequence.

Input Specification

The input consists of one or more test cases. Each test case consists of one integer n with $1 \leq n \leq 5842$. Input is terminated by a value of zero (0) for n.

Output Specification

For each test case, print one line saying `The nth humble number is number.` Depending on the value of n, the correct suffix "st", "nd", "rd", or "th" for the ordinal number n-th has to be used like it is shown in the sample output.

Sample Input

```
1
2
3
4
11
12
13
21
22
23
100
1000
5842
0
```

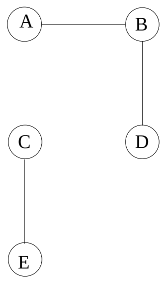
Sample Output

```
The 1st humble number is 1.
The 2nd humble number is 2.
The 3rd humble number is 3.
The 4th humble number is 4.
The 11th humble number is 12.
The 12th humble number is 14.
The 13th humble number is 15.
The 21st humble number is 28.
The 22nd humble number is 30.
The 23rd humble number is 32.
```

```
The 100th humble number is 450.
The 1000th humble number is 385875.
The 5842nd humble number is 2000000000.
```

4.2 Composantes connexes

Exercice 5 Graph Connectivity. UVa : 459.
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 and Output

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The first line of each input set 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. The sample input section contains a possible input set for the graph pictured above.

Input is terminated by a blank line.

For each test case, output the number of maximal connected subgraphs. The outputs of two consecutive cases will be separated by a blank line.

Sample Input

```
1

E
AB
CE
DB
EC
```

Sample Output

```
2
```