The Input

The input consists of a sequence of integer pairs *n* and *p* with each integer on a line by itself. For all such pairs $1 \leq n \leq 200$, $1 \leq p < 10^{101}$ and there exists an integer *k*, $1 \leq k \leq 10^9$ such that $k^n = p$.

The Output

For each integer pair *n* and *p* the value $\sqrt[n]{p}$ should be printed, i.e., the number *k* such that $k^n = p$.

Sample Input

```
2
16
3
27
7
4357186184021382204544
```

Sample Output

```
4
3
1234
```

Roman Roulette

The historian Flavius Josephus relates how, in the Romano-Jewish conflict of 67 A.D., the Romans took the town of Jotapata which he was commanding. Escaping, Jospehus found himself trapped in a cave with 40 companions. The Romans discovered his whereabouts and invited him to surrender, but his companions refused to allow him to do so. He therefore suggested that they kill each other, one by one, the order to be decided by lot. Tradition has it that the means for effecting the lot was to stand in a circle, and, beginning at some point, count round, every third person being killed in turn. The sole survivor of this process was Josephus, who then surrendered to the Romans. Which begs the question: had Josephus previously practised quietly with 41 stones in a dark corner, or had he calculated mathematically that he should adopt the 31st position in order to survive?

Having read an account of this gruesome event you become obsessed with the fear that you will find yourself in a similar situation at some time in the future. In order to prepare yourself for such an eventuality you decide to write a program to run on your hand-held PC which will determine the position that the counting process should start in order to ensure that you will be the sole survivor.

In particular, your program should be able to handle the following variation of the processes described by Josephus. $n > 0$ people are initially arranged in a circle, facing inwards, and numbered from 1 to n. The numbering from 1 to n proceeds consecutively in a clockwise direction. Your allocated number is 1. Starting with person number i, counting starts in a clockwise direction, until we get to person number k ($k > 0$), who is promptly killed. We then proceed to count a further k people in a clockwise direction, starting with the person immediately to the left of the victim. The person number k so selected has the job of burying the victim, and then returning to the position in the circle that the victim had previously occupied. Counting then proceeeds from the person to his immediate left, with the kth person being killed, and so on, until only one person remains.

For example, when $n = 5$, and $k = 2$, and $i = 1$, the order of execution is 2, 5, 3, and 1. The survivor is 4.

Input and Output

Your program must read input lines containing values for n and k (in that order), and for each input line output the number of the person with which the counting should begin in order to ensure that you are the sole survivor. For example, in the above case the safe starting position is 3. Input will be terminated by a line containing values of 0 for n and k.

Your program may assume a maximum of 100 people taking part in this event.

Sample Input

```
1 1
1 5
0 0
```

Sample Output

```
1
1
```

Joseph's Cousin

The Problem

The Joseph's problem is notoriously known. For those who are not familiar with the problem, among n people numbered 1,2…n, standing in circle every mth is going to be executed and only the life of the last remaining person will be saved. Joseph was smart enough to choose the position of the last remaining person, thus saving his life to give the message about the incident.

Although many good programmers have been saved since Joseph spread out this information, Joseph's Cousin introduced a new variant of the malignant game. This insane character is known for its barbarian ideas and wishes to clean up the world from silly programmers. We had to infiltrate some the agents of the ACM in order to know the process in this new mortal game. In order to save yourself from this evil practice, you must develop a tool capable of predicting which person will be saved.

The Destructive Process

The persons are eliminated in a very peculiar order; m is a dynamical variable, which each time takes a different value corresponding to the prime numbers' succession (2,3,5,7…). So in order to kill the ith person, Joseph's cousin counts up to the ith prime.

The Input

It consists of separate lines containing n [1..3501], and finishes with a 0.

The Output

The output will consist in separate lines containing the position of the person which life will be saved.

Sample Input

```
6
0
```

Sample Output

```
4
```

Integer Inquiry

One of the first users of BIT's new supercomputer was Chip Diller. He extended his exploration of powers of 3 to go from 0 to 333 and he explored taking various sums of those numbers.

``This supercomputer is great," remarked Chip. ``I only wish Timothy were here to see these results." (Chip moved to a new apartment, once one became available on the third floor of the Lemon Sky apartments on Third Street.)

Input

The input will consist of at most 100 lines of text, each of which contains a single VeryLongInteger. Each VeryLongInteger will be 100 or fewer characters in length, and will only contain digits (no VeryLongInteger will be negative).

The final input line will contain a single zero on a line by itself.

Output

Your program should output the sum of the VeryLongIntegers given in the input.

Sample Input

```
123456789012345678901234567890
123456789012345678901234567890
123456789012345678901234567890
0
```

Sample Output

```
370370367037037036703703703670
```

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.

Input and Output

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.

Sample Input

```
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
```

Sample Output

```
*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.
```

APPENDIX B
COMMON CODES/ROUTINES FOR PROGRAMMING

This part of this book contains some COMMON codes/routines/pseudocodes for programmers that one can EASILY use during the contest hours.[10]

Finding GCD,LCM and Prime Factor

```
#include<iostream>
#include<vector>
#include<fstream>
using namespace std;

int gcd(int a, int b)
{
int r=1;
if(a>b)
 {
 while ( r!=0)
  {
  r=a%b;
  a=b;
  b=r;
 }
return a;
}
else if ( b>a)
 {
 while (r!=0)
  {
   r=b%a;
   b=a;
   a=r;
  }
return b;
}
else if ( a==b)
  return a;
}

int lcm(int a, int b)
{
return a*b/gcd(a,b);
}

bool isprime(int a)
{
if(a==2) return true;
else
{
  for(int i=2;i<a/2+1;++i)
  if(a%i==0) return false;
  if(i==a/2) return true;}
} }
```

```
vector<int> primefactors(int a)
{
 vector<int> primefactors;
 for(int i=1;i<=a;++i)
 if(a%i==0&&isprime(i))
 {
  primefactors.push_back(i);
  a=a/i;i=1;
 }
 return primefactors;
}

int main()
{
int a=5,b=2500;
out<<"GCD = "<<gcd(a,b)<<endl;
out<<"LCM = "<<lcm(a,b)<<endl;
if(!isprime(b)) out<<"not
prime"<<endl;
vector <int> p =
primefactors(b);
for(int i=0;i<p.size();++i)
out<<p[i]<<endl;
return 0;
}
```

Recursively find the GCD by Euclid's Algorithm

```
//******        GCD using recursive way          **********************/
#include<stdio.h>
unsigned int Euclid_g_c_d(unsigned int a, unsigned int b){
if(b==0)
            return a;
     else
            return Euclid_g_c_d(b,a%b);
}
int main(){
     unsigned int a,b,gcd;
     while( scanf("%d%d",&a,&b)==2){
            gcd=Euclid_g_c_d(a,b);
            printf("%d",gcd);
     }
     return 0;
}
```

The another application of GCD is Extended Euclid's Algorithm..We can solve the formula **d = ax + by => gcd(a,b) = ax + by.** This algorithm take the inputs a and b..and out put the d(gcd of a,b) and the value of the root x and y.

```
//******    Extended Euclid's Algorithm      *****************//
#include<stdio.h>
unsigned long d,x,y;
void Extended_Euclid(unsigned long a, unsigned long b){
     unsigned long x1;
     if(b>a){
            x1=a;   //if b>a so I used this if condition
            a=b;   // result is ok but x and y swaped
            b=x1;
     }
     if(b==0){
            d=a;
            x=1;
            y=0;
            return;
     }
     Extended_Euclid(b,a%b);
     d = d;
     x1 = x-(a/b) * y;
     x = y;
     y = x1;
}
int main(){
     unsigned long a,b;       //   d = gcd(a,b) = ax+by
            while(scanf("%lu %lu", &a, &b)==2){
            Extended_Euclid(a,b);
            printf("%lu %lu %lu\n",d,x,y);}return 0;}
```

GCD of N numbers:

This program takes a number that is for how many numbers you want to find the GCD and show the GCD of them. This takes input until end of file.

```
//********            GCD for n numbers            ******************//
#include<stdio.h>
int main(){
     long gcd,a,b,n,i;
     while(scanf("%ld",&n)==1){
            scanf("%ld%ld",&a,&b);
            i=2;
            while(i<n){
                   gcd=g_c_d(a,b);
                                        a=gcd;



i++;
                   scanf("%ld",&b);
            }
            gcd=g_c_d(a,b);
            printf("%ld\n",gcd);
     }
     return 0;
}
```

LCM (Lowest Common Multiple)

It is also a important topic in programming. Suppose different clocks are set in 12 point, and each and every clock has their own time to complete the cycle. When every clock again set in the 12 point.? This type of problem can be solved by LCM. LCM was implemented by the help of GCD.

```
LCM (m, n) = [m / GCD(m, n)] * n;
Or,  D1=m / GCD(m, n);
     D2=n  / GCD(m, n);
     LCM(m, n)  =  D1 * D2 * GCD(m, n);
```

Here is the code of some int numbers LCM.

```
long GCD(long a, long b){
     if(b==0)
            return a;
     else
            return GCD(b,a%b);}
long LCM(long a,long b){
```

```
     return (a/GCD(a,b)*b);
}
int main(){
     long a,b;
     int n,i;
     while(scanf("%d",&n)==1){
            if(n==0) break;
            scanf("%ld",&a);
            for(i=1;i<n;i++){
                  scanf("%ld",&b);
                  a=LCM(a,b);
            }
            printf("%ld\n",a);
     }
     return 0;}
```

Factorials

n!= n * (n-1) * (n-2) * (n-3)……upto 1.
5!= 5 * (5-1) * (5-2) * (5-3) * 2 * 1 = 120.
This is a recursion process.

```
//***                Factorial by recurtion            *************//
#include<stdio.h>
long fac(long x){
     if(x==0)
            return 1;
     return x * fac(x-1);
}
/*****************************************************************/
int main(){
     long n,res;
     while(scanf("%ld",& n)==1){
            res=fac(n);
            printf("%ld\n",res);
     }
     return 0;}
```

This is very easy, but one thing we have to know that factorials increases multiply. The 12^{th} factorial is 479001600 which is very big so can you imagine the factorial of 1000!….? The above program can compute upto 12^{th} factorials so for less then 12^{th} we can use it. But for time consideration in ACM we always try to avoids the recursion process.

Iterative Factorial

```
//*********    Factorial by normal multiplication   *********//
#include<stdio.h>
long fac(long x){
     long i,res=1;
     for(i=1;i<=x;i++)
            res*=i;
     return res;
}
int main(){
     long n,res;
     while(scanf("%ld",&n)==1){
            res=fac(n);
            printf("%ld\n",res);
     }
     return 0;
}
```

So, how can we compute the factorial of big numbers? Here is a code that can show the factorials up to 1000. And this a ACM problem (623-500!). This code is very efficient too!. We pre-calculate all the 1000^{th} factorials in a two dimensional array. In the main function just takes input and then just print.

Factorial of Big Number!

```
#include<stdio.h>
#include<string.h>
char f[10000];
char factorial[1010][10000];
void multiply(int k){
     int cin,sum,i;
     int len = strlen(f);
     cin=0;
     i=0;
     while(i<len){
            sum=cin+(f[i] - '0') * k;
            f[i] = (sum % 10) + '0';
            i++;
            cin = sum/10;
     }
     while(cin>0){
            f[i++] = (cin%10) + '0';
            cin/=10;
     }
     f[i]='\0';
     for(int j=0;j<i;j++)
            factorial[k][j]=f[j];
     factorial[k][i]='\0';}
```

```
void fac(){
      int k;
      strcpy(f,"1");
      for(k=2;k<=1000;k++)
            multiply(k);
}
void print(int n){
      int i;
      int len = strlen(factorial[n]);
      printf("%d!\n",n);
      for(i=len-1;i>=0;i--){
            printf("%c",factorial[n][i]);
      }
      printf("\n");
}
int main(){
      int n;
      factorial[0][0]='1';
      factorial[1][0]='1';
      fac();
      while(scanf("%d",&n)==1){
            print(n);
      }
      return 0;
}
```

Factorial Frequencies

Now how many digits in the factorial of N numbers..? Yes we can count them by the above process. But we can do it simply!

```
//*********    How many digits in N factorials    ******************//
#include<stdio.h>
#include<math.h>
/*****************************************************************/
double count_digit(long a){
      double sum;
      long i;
      if(a==0) return 1;
      else{
            sum=0;
            for(i=1;i<=a;i++){
                  sum+=log10(i);
            }
            return floor(sum)+1;
      }
}
/*****************************************************************/
int main(){
      double sum;
      long n;
```

```
      while(scanf("%ld",&n)==1){
            sum=count_digit(n);
            printf("%.0lf\n",sum);
      }
      return 0;
}
```

How many trailing zeros in Factorials ?

We know that factorials of any number has so many zeros (0's) at last…example 17! = 355687428096000. It has 3 last zero digits. So we now see how many trailing zeros have in N factorials.

```
//***************          How many trailing zeros in N!
**********/*******//
#include<stdio.h>
long zero(long number,long factor){
      long total,deno;
      if(number==5) return 1;
      total=0;
      deno=factor;
      while(deno<number){
            total+=number/deno;
            deno*=factor;
      }
      return total;
}
/*****************************************************************/
int main(){
      long N,c2,c1;
      while(scanf("%ld",&N)==1){
            c1=zero(N,2);
            c2=zero(N,5);
            if(c1<c2) printf("%ld\n",c1);
            else printf("%ld\n",c2);
      }
      return 0;
}
```

Big Mod

We are not always interested in the full answers, however. Sometimes the remainder suffices for our purposes. In that case we use Modular Arithmetic. The key to efficient modular arithmetic is understanding how the basic operation of addition, subtraction, and multiplication work over a given modulus.

- Addition:- (x + y)mod n…..? = ((x mod n) + (y mod n)) mod n.
- Subtraction:- (x-y) mod n....? = ((x mod n) - (y mod n)) mod n.

We can convert a negative number mod n to positive number by adding a multiple of n to it.

- Multiplication:- xy mod n…? = (x mod n)(y mod n) mod n.

What is x^y mod n…? Exponentiation is just repeated multiplication, $(x \bmod n)^y \bmod n$. So if the x and y are very big equal to 32 bit. Then we can also compute it without overflow.

```
/* Big mod */
#include<stdio.h>
long square(long s){
    return s*s;
}
long bigmod(long b, long p, long m){
  if (p == 0)
    return 1;
  else if (p%2 == 0)
    return square( bigmod (b,p/2,m)) % m; // square(x) = x * x
  else
    return ((b % m) * bigmod( b,p-1,m)) % m;
}
int main(){
    long b,p,m,sum;
    while(scanf("%ld%ld%ld",& b,& p,& m)==3){
         sum = bigmod( b, p, m);
         printf("%ld\n", sum);
    }
    return 0;}
```

Number Conversions - Integer ⇔ Binary

Convert an integer number into binary number

```
int inttobin(unsigned int a,int
*bin)
{
      int one=0;
      unsigned int c=1;
      int i ;
      for(i=31;i>=0;i--)
      {
            one = one + (a&c);
            bin[i] = (a&c)?1:0;
            c<<=1;
      }
      return one; }
```

Convert a binary number into binary number

```
unsigned int creatnum(int *num)
{
      int i;
      unsigned int a=0;
      for(i=0;i<32;i++)
      {      a= a|num[i];
             if(i!=31)

a <<= 1;
      }
      return a;
}
```

Integer ⇔ hex ⇔ Integer

```
char hex[100];
int num = 4095;
sprintf(hex,"%X",num);// convert the num in upper case hex decimal in hex
sprintf(hex,"%x",num);// convert the num in lower case hex decimal in hex
sscanf(hex,"%x",&num);// convert the hex number hex in integer num
```

Integer to any base

Following code help you to convert any integer number on any base from 2 to 36.

```
#include<string.h>
#include<stdio.h>
#define MAXDGT 200       /* maximum number of digit */

/* ********************************************************     */
/*     A general swap function that swap two object.            */
/*     Input  : Two object.                                  */
/*     Return : None                                         */
/* ********************************************************     */
template <class T>

void swap(T &x, T &y)
{   T tmp=x;   x=y;   y=tmp;    };

/************************************************************* */
/*     A general string reverse function that reverse the       */
/*     passed string and store the string int to the parameter.  */
/*     Input  : A string                                     */
/*     Return : None                                            */
/************************************************************* */
void revstr(char *str)
{    int i=0,l=strlen(str);
     while(i<l)
           swap(str[i++],str[--l]);
}

/*************************************************************** */
/*     A general base conversion function                        */
/*     Input  : A number n and a base b                             */
/*     Return : A character array which contain the number n in base b */
/* ********************************************************************* */
char *itob(long int n,int b=10)
{    char num[MAXDGT];
     int j,sign;
     register int i=0;
```

```
     if( (sign=n) <0 )
           n= -n;
do    {     j=n%b;
                num[i++]=(j<10) ? (j+'0'): ('A'+j-10);
           }while((n/=b)!=0);

     if(sign < 0)        num[i++]='-';

     num[i]='\0';
     revstr(num);
     return num;
}

/* Sample main */
main(void)
{    printf(itob(71],36));      }
```

Decimal To Roman

Roman number is very useful number system. Often we need to convert a decimal number into Roman number. In Roman number the following symbol are used.

i, v, x, l, c, m for 1, 5, 10, 50, 100 and 500 respectively.

```
#include<stdio.h>
```

```
/* ********************************    */
/*     Convert number 1 to 10          */
/*     Input  : A integer number       */
/*     Return : None                   */
/* ********************************    */

void unit(int n)
{      switch(n){
        case 3 :  printf("i");
        case 2 :  printf("i");
        case 1 :  printf("i"); break;
        case 4 :  printf("i");
        case 5 :  printf("v"); break;
        case 6 :  printf("vi"); break;
        case 7 :  printf("vii");  break;
        case 8 :  printf("viii"); break;
        case 9 :  printf("ix");  break;
}
}

/* ********************************    */
/*     Convert number 10 to 100        */
```

```
/*      Input  : A integer number          */
/* ********************************    */
void ten(int n)

{       switch(n){
         case 3 :  printf("x");
         case 2 :  printf("x");
         case 1 :  printf("x"); break;
         case 4 :  printf("x");
         case 5 :  printf("l"); break;
         case 6 :  printf("lx"); break;
         case 7 :  printf("lxx");  break;
         case 8 :  printf("lxxx"); break;
         case 9 :  printf("xc");  break;
 }
}

/* ********************************    */
/*      Convert number 100 to 500          */
/*      Input  : A integer number          */
/* ********************************    */

void hnd(int n)
{       switch(n){
         case 3 :  printf("c");
         case 2 :  printf("c");
         case 1 :  printf("c"); break;
         case 4 :  printf("c");
         case 5 :  printf("M"); break;
 }
}

/* ************************************************      */
/*      Convert an integer number into roman system       */
/*      Input  : A integer number                         */
/* ************************************************      */
void roman(int n)
{       int a,i;
        if(n>=500)
                {       a=n/500 ;
                        for(i=1;i<=a;i++)
                                printf("M");
                }
                n=n%500;
                hnd(n/100);
                n=n%100;
                ten(n/10);
                unit(n%10);
} main(void)
{
roman(390);
return 0;
}
```

Josephus Problem

Flavius Josephus was a famous historian of the first century. Legend has it that Josephus wouldn't have lived to become famous without his mathematical talents. During the Jewish-Roman war, he was among a band of 41 Jewish rebels trapped in cave by the Romans.Preferring suicide to capture, the rebels decided to form a circle and, proceeding around it, to kill every third remaining person until no one was left. But Josephus, along with an un-indicted co-conspirator, wanted none of this suicide nonsense; so he quickly calculated where he and his friend stand in the vicious circle.

```
#include <stdio.h>
#define MAX 150

/** Implementaiom of circular queue  **/


int queue[MAX];
int e,f,r;

/** _f is front; _r is rear & _e is element number **/

/* ********************************************** */
/*   A general push algorithm act over queue    */
/* ********************************************** */

void push(int i)
{
   r++;
   e++;
   if(r>=MAX)
      r=1;
   queue[r] = i;
}


/* ********************************************** */
/*   A general pop algorithm act over queue     */
/* ********************************************** */

int pop(void)
{
   e--;
   int i=queue[f];
   f++;
   if(f>=MAX)
      f = 1;
   return i;
}
```

```
/** End of circular queue **/
/* ********************************    */
/* A general joshusf function.           */
/* This function start removing          */
/* element from first element            */
/* and return the sirviver              */
/* ********************************   */

int joshups(int n,int v)
{
   register int i;
   e=n;
   for(i=1;i<=n;i++)
      queue[i] = i;
   f = 1; // 0 for serviving first element.
   r = n; // n+1 for serviving first element.
   i = 0;
   if(n > 1)
      pop();
   while(e!=1)
   {
      if(i!=v)
      {
         i++;
         push(pop());
      }
      else
      {
             pop();
             i = 0;
      }
   }
   return queue[f];
}


/*sample main function*/

main(void)
{
   int i,m;
   scanf("%d",&i);
   while(i)
   {   m=1;
      while((joshups(i,m++)) != 13 );
      printf("%d",m);
      putchar('\n');
      scanf("%d",&i);
   }

   return 0; }
```

Combinations

```
/* ******************************************** */
/*        Calculate nCm                          */
/*        Input : Two integer number n m         */
/*        Return : The nCm                       */
/* ******************************************** */

double nCr(int n,int m)
{
  int k;
  register int i,j;
  double c,d;

  c=d=1;
  k=(m>(n-m))?m:(n-m);
  for(j=1,i=k+1;(i<=n);i++,j++)
  {
      c*=i;
      d*=j;
      if( !fmod(c,d)  && (d!=1) )
      {   c/=d;
          d=1;
      }
  }

  return c;
}

/* A sample main function */
main(void)
{
  int n,m;

  while(scanf("%d%d",&n,&m)!=EOF)
  printf("%.0lf\n",nCr(n,m));
  return 0;

}
```

Another way to calculate the nCm by using the Pascal's triangel.

```
#include<stdio.h>
#define MAXTRI 50

unsigned long int pas[MAXTRI][MAXTRI];

void pascals(int n)
{
```