Univerza v Ljubljani

Prvenstvo UNIVERZE V LJUBLJANI v programiranju

2002

poskusno tekmovanje

Fakulteta za matematiko in fiziko

INNOVATIVE SOFTWARE SOLUTIONS
HERMES SoftLab

Pokrovitelji:

Prvenstvo v programiranju 2002

poskusno tekmovanje 13. maj 2002

Prvenstvo v programiranju 2002

poskusno tekmovanje 13. maj 2002

NAVODILA

Prijava: Prijavite se lahko le na en računalnik. Če bodo med tekmovanjem z njim slučajno težave, javite vodstvu tekmovanja. V primeru, da boste hkrati prijavljeni na dveh računalnikih, bo ekipa diskvalificirana.

Uporabniško ime je sestavljeno iz besede acm in številke računalnika.

Primer: uporabniško ime na računalniku št. 1 je: acm01

uporabniško ime na računalniku št. 12 je: acm12

Geslo je sporočeno pred začetkom tekmovanja.

Vaše domače področje na disku je `/home/acmXX` kjer je XX številka računalnika

Priprava delovnega okolja

Po uspešni prijavi morate odpreti dva terminala in program Netscape.

Terminal št.1: v njem boste pisali programe, prevajali programe in pošiljali programe sodniku.

Terminal št.2: v njem boste dobivali obvestila o pravilnosti rešitev in pošiljali morebitna vprašanja sodniku.

Program Netscape: bo omogočal pregled trenutne uspešnosti ekip

Pisanje programov:

Na voljo so urejevalniki besedil: *vi, joe, jed, mcedit, emacs*, *gedit in kedit*

Ime programov je točno določeno pri opisu nalog. Pazite na `return 0,` ki mora zaključiti vsak uspešno izveden C program.

Prevajanje programov:

Za prevajanje programov morate uporabljati naslednje prevajalnike:

```
C      (*.c)        gcc    primer:   gcc -o program program.c
C++    (*.cpp)      g++    primer:   g++ -o program program.cpp
Pascal (*.pas)      gpc    primer:   gpc -o program program.pas
```

Pošiljanje programov:

Za pošiljanje programov sodniku uporabljate ukaz `submit`.

Primer: `submit program.c`

Sporočila o napakah – PE in WA

Zaradi različnih razlogov je zelo težko razlikovati med `Presentation Error` in `Wrong Answer`. Ker preverjanje poteka več ali manj avtomatsko, program pogosto ne zna razlikovati med tem, ali ste le pozabili piko v odgovoru, ali pa ste napačno zračunali. Zato tudi `Wrong Answer` včasih lahko pomeni le manjkajoč znak v odgovoru. Praviloma naj bi sporočilo `PE` dobili le v primeru napačno postavljenih presledkov ali ločil in praznih vrstic, ni pa to nujno!

Komunikacija sodniki – ekipa:

V terminalu 2 izvedete ukaz: `telnet svarun`

Za prijavo na računalnik svarun uporabite isto uporabniško ime in geslo, kot ste ga uporabili za prijavo na vaš računalnik. V tem terminalskem oknu se bodo izpisovala sodnikova sporočila vam. Če želite poslati sporočilo sodniku, morate v tem terminalu napisati ukaz:

```
write sodnik
```

in potem sporočilo, ki ga zaključite s kombinacijo tipk CTRL + D na začetku prazne vrstice. Prosimo, da **ne** pošiljate sporočil tipa `... Sedaj smo poslali program ...` in `... Naš program zagotovo dela, pa dobimo WA ...` Ekipa, ki bo pošiljala tovrstna sporočila, je lahko tudi diskvalificirana.

Trenutni rezultati:

Program Netscape: bo omogočal pregled trenutne uspešnosti ekip

Po zagonu programa greste na naslov:

http://svarun.fmf.uni-lj.si/index.html

Kadar želite videti trenutno stanje, morate narediti »reload«.

Pritožbe:

So možne le takoj po zaključku tekmovanja. Kasneje se rezultati lahko spremenijo le v primeru drastične sodniške napake.

Preklapljanje med angleško in slovensko tipkovnico:

(Velja samo za okolje X windows)

Če po prijavi v okolje X windows desno spodaj ne vidite sličice za izbiro tipkovnice, morate narediti sledeče:

V K-meniju izberete:

`KDE Control Center -> Input Device -> International Keyboard`

In dodaste hrvaško tipkovnico (ker slovenske še ni), potem izberete še zavihek `Startup` in izberete obe možnosti: `Autostart` in `Docked`

Odjavite se iz X okolja in po ponovni prijavi boste desno spodaj imeli možnost izbire tipkovnice.

Izbrana tipkovnica potem velja za celotno X okolje.

Gondwanaland Telecom

Program TELECOM.C, TELECOM.CPP, TELECOM.PAS

Gondwanaland Telecom makes charges for calls according to distance and time of day. The basis of the charging is contained in the following schedule, where the charging step is related to the distance:

Charging Step (distance)	Day Rate 8am to 6pm	Evening Rate 6pm to 10pm	Night Rate 10pm to 8am
A	0.10	0.06	0.02
B	0.25	0.15	0.05
C	0.53	0.33	0.13
D	0.87	0.47	0.17
E	1.44	0.80	0.30

All charges are in dollars per minute of the call. Calls which straddle a rate boundary are charged according to the time spent in each section. Thus a call starting at 5:58 pm and terminating at 6:04 pm will be charged for 2 minutes at the day rate and for 4 minutes at the evening rate. Calls less than a minute are not recorded and no call may last more than 24 hours.

Write a program that reads call details and calculates the corresponding charges.

Input and Output

Input lines will consist of the charging step (upper case letter 'A'..'E'), the number called (a string of 7 digits and a hyphen in the approved format) and the start and end times of the call, all separated by exactly one blank. Times are recorded as hours and minutes in the 24 hour clock, separated by one blank and with two digits for each number. Input will be terminated by a line consisting of a single #.

Output will consist of the called number, the time in minutes the call spent in each of the charge categories, the charging step and the total cost in the format shown below.

Sample input

```
A 183-5724 17 58 18 04
#
```

Sample output

```
183-5724 2 4 0 A 0.44
```

Factors and Factorials

Program FACTORS.C, FACTORS.CPP, FACTORS.PAS

The factorial of a number N (written N!) is defined as the product of all the integers from 1 to N. It is often defined recursively as follows:

$$1! = 1$$

$$N! = N * (N - 1)!$$

Factorials grow very rapidly -- 5! = 120, 10! = 3628800. One way of specifying such large numbers is by specifying the number of times each prime number occurs in it, thus 825 could be specified as (0 1 2 0 1) meaning no twos, 1 three, 2 fives, no sevens and 1 eleven.

Write a program that will read in a number N (2 <= N <= 100) and write out its factorial in terms of the numbers of the primes it contains.

Input

Input will consist of a series of lines, each line containing a single integer N. The file will be terminated by a line consisting of a single 0.

Output

Output will consist of a series of blocks of lines, one block for each line of the input. Each block will start with the number N, right justified in a field of width 3, and the characters '!', space, and '='. This will be followed by a list of the number of times each prime number occurs in N!.

These should be right justified in fields of width 3 and each line (except the last of a block, which may be shorter) should contain fifteen numbers. Any lines after the first should be indented. Follow the layout of the example shown below exactly.

Sample input

```
5
53
0
```

Sample output

```
  5! =  3  1  1
 53! = 49 23 12  8  4  4  3  2  2  1  1  1  1  1  1
        1
```

The MTM Machine

Program MTM.C, MTM.CPP, MTM.PAS

The MTM is one of the first digital machines ever designed. The aim of the machine is to process positive integer numbers, but due to the primitive nature of the machine only some numbers are accepted for processing; such numbers are called *acceptable*. When a number is accepted by MTM, the machine outputs another number, according to the rules stated below. When a number is not accepted, the machine simply outputs `NOT ACCEPTABLE`.

A number is a non empty string of decimal digits. Given two numbers N and M, when we write NM we mean the number formed by the digits of N followed immediately by the digits of M. For example, if N is 856 and M is 112 then NM is 856112. For any number X, the associate of X is the number $X2X$. For example, the associate of 78 is 78278.

We say that a number X *produces* a number Y, if number X is acceptable and when given as input to the machine MTM, the number returned by the machine is Y.

The behaviour of the MTM machine is governed by the following rules:

Rule 0:

A number containing the digit 0 (zero) is not acceptable.

Rule 1:

Given any number X not containing a digit zero, then number $2X$ produces X. For example, 234 produces 34.

Rule 2:

Given any pair of numbers X, Y, if X produces Y then $3X$ produces the associate of Y. For example, 25 produces 5 by Rule 1, so 325 produces 525.

Rule 3:

No other numbers are acceptable.

Your task here is to write a program that simulates the MTM machine.

Input

The input file contains a set of test cases. Each test case appears in a separate line, and consists of a single positive number N, $N < 10^{32}$, to be processed by the MTM machine. The file ends with a line containing the number 0 that should not be processed.

You may assume that the largest number output by the machine has at most 1000 digits.

Output

For each test case, your program should write one line with the output produced by the machine if the corresponding number is acceptable; otherwise your program should write `NOT ACCEPTABLE`.

Sample Input

```
20
22
42
32
33289
0
```

Sample Output

```
NOT ACCEPTABLE
2
NOT ACCEPTABLE
NOT ACCEPTABLE
89289289289
```

Greedy Gift Givers

Program GIFT.C, GIFT.CPP, GIFT.PAS

The Problem

This problem involves determining, for a group of gift-giving friends, how much more each person gives than they receive (and vice versa for those that view gift-giving with cynicism).

In this problem each person sets aside some money for gift-giving and divides this money evenly among all those to whom gifts are given.

However, in any group of friends, some people are more giving than others (or at least may have more acquaintances) and some people have more money than others.

Given a group of friends, the money each person in the group spends on gifts, and a (sub)list of friends to whom each person gives gifts; you are to write a program that determines how much more (or less) each person in the group gives than they receive.

The Input

The input is a sequence of gift-giving groups. A group consists of several lines:

- the number of people in the group,
- a list of the names of each person in the group,
- a line for each person in the group consisting of the name of the person, the amount of money spent on gifts, the number of people to whom gifts are given, and the names of those to whom gifts are given.

All names are lower-case letters, there are no more than 10 people in a group, and no name is more than 12 characters in length. Money is a non-negative integer less than 2000.

The input consists of one or more groups and is terminated by a group of size 0.

The Output

For each group of gift-givers, the name of each person in the group should be printed on a line followed by the net gain (or loss) received (or spent) by the person. Names in a group should be printed in the same order in which they first appear in the input.

The output for each group should be separated from other groups by a blank line. All gifts are integers. Each person gives the same integer amount of money to each friend to whom any money is given, and gives as much as possible. Any money not given is kept and is part of a person's ``net worth'' printed in the output.

Sample Input

```
5
dave laura owen vick amr
dave 200 3 laura owen vick
owen 500 1 dave
amr 150 2 vick owen
laura 0 2 amr vick
vick 0 0
3
liz steve dave
liz 30 1 steve
steve 55 2 liz dave
dave 0 2 steve liz
0
```

Sample Output

```
dave 302
laura 66
owen -359
vick 141
amr -150

liz -3
steve -24
dave 27
```

Frogger

FROGGER.C FROGGER.CPP FROGGER.PAS

Freddy Frog is sitting on a stone in the middle of a lake. Suddenly he notices Fiona Frog who is sitting on another stone. He plans to visit her, but since the water is dirty and full of tourists' sunscreen, he wants to avoid swimming and instead reach her by jumping.

Unfortunately Fiona's stone is out of his jump range. Therefore Freddy considers to use other stones as intermediate stops and reach her by a sequence of several small jumps.

To execute a given sequence of jumps, a frog's jump range obviously must be at least as long as the longest jump occuring in the sequence.

The *frog distance* (humans also call it *minimax distance*) between two stones therefore is defined as the minimum necessary jump range over all possible paths between the two stones.

You are given the coordinates of Freddy's stone, Fiona's stone and all other stones in the lake. Your job is to compute the frog distance between Freddy's and Fiona's stone.

Input Specification

The input file will contain one or more test cases. The first line of each test case will contain the number of stones n ($2 <= n <= 200$). The next n lines each contain two integers x_i, y_i ($0 <= x_i, y_i <= 1000$) representing the coordinates of stone #i. Stone #1 is Freddy's stone, stone #2 is Fiona's stone, the other n-2 stones are unoccupied. There's a blank line following each test case. Input is terminated by a value of zero (0) for n.

Output Specification

For each test case, print a line saying ``Scenario #$x$'' and a line saying ``Frog Distance = y'' where x is replaced by the test case number (they are numbered from 1) and y is replaced by the appropriate real number, printed to three decimals. Put a blank line after each test case, even after the last one.

Sample Input

```
2
0 0
3 4

3
17 4
19 4
18 5
```

```
0
```

Sample Output

```
Scenario #1
Frog Distance = 5.000

Scenario #2
Frog Distance = 1.414
```