Odprto ekipno prvenstvo UNIVERZE V LJUBLJANI v programiranju

Odprto ekipno prvenstvo UNIVERZE V LJUBLJANI v programiranju 2003

2003

3. krog

Glavni pokrovitelji

PARSEK

Microsoft®

UNIVERZA V LJUBLJANI

Fakulteta za matematiko in fiziko

Pokrovitelji

anni

atlantis
V SREDIŠČU INFORMACIJSKEGA SPLETA

avtenta.si

halcom

PCFormat

Perftech

Panda Software
Protivirusne rešitve

MARAND
Napredna računalniška hiša

mlacom

IGEA

IBM

ORIA
COMPUTERS

Joker

Šolski servis
Janez Vrankar

Monitor

DATALAB
SPREMENITE PODATKE V DOBIČEK!

dMFA
ZALOŽNIŠTVO

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, kwrite, 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
Java	(*.java)	gcj	primer:	gcj –o program ––main=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: `ssh svarun.fmf.uni-lj.si`

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:

Z ukazom v terminalu:

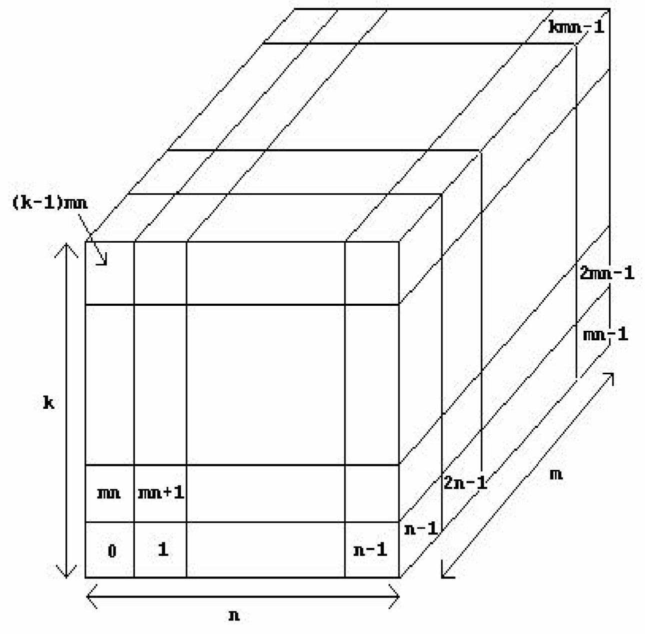
```
sexkbmap si
setxkbmap us
```

Nastavitve veljajo za novo odprte programe.

Space Station Shielding

Program shielding.c, shielding.cpp, shielding.java, shielding.pas

Roger Wilco is in charge of the design of a low orbiting space station for the planet Mars. To simplify construction, the station is made up of a series of Airtight Cubical Modules (ACM's), which are connected together once in space. One problem that concerns Roger is that of (potentially) lethal bacteria that may reside in the upper atmosphere of Mars. Since the station will occasionally through the upper atmosphere, it is imperative that extra shielding be used on all faces of the ACMs which make up the external surface of the station. Roger has made certain that where two ACMs touch, either edge-to-edge or face-to-face, that joint is sealed so no bacteria can sneak through. Any face of an ACM shared by another ACM will not need shielding, of course, nor will a face which cannot be reached from the outside. Roger could just put extra shielding on all of the faces of every ACM, but the cost would be prohibitive. Therefore, he wants to know the exact number of ACM faces which need the extra shielding.



Input

Input consists of multiple problem instances. Each instance consists of a specification of a space station. We assume that each space station can fit into an *n* x *m* x *k* grid ($1 \le n, m, k \le 60$), where each grid cube may or may not contain an ACM. We number the grid cubes 0, 1, 2,..., *kmn*-1 as shown in the diagram below. Each space station specification then consists of the following: the first line contains four positive integers *n m k l*, where *n, m* and *k* are as described above and l is the number of ACM's in the station. This is followed by a set of lines which specify the l grid locations of the ACM's. Each of these lines contain 10 integers (except possibly the last). Each space station is fully connected (i.e., an astronaut can move from one ACM to any other ACM in the station without leaving the station). Values of $n = m = k = l = 0$ terminate input.

Output

For each problem instance, you should output one line of the form

```
The number of faces needing shielding is s.
```

where s is for you to determine.

Sample Input

```
2 2 1 3
0 1 3
3 3 3 26
0 1 2 3 4 5 6 7 8 9
10 11 12 14 15 16 17 18 19 20
21 22 23 24 25 26
0 0 0 0
```

Sample Output

```
The number of faces needing shielding is 14.
The number of faces needing shielding is 54.
```

Traffic Lights

Program traffic.c, traffic.cpp, traffic.java, traffic.pas

One way of achieving a smooth and economical drive to work is to 'catch' every traffic light, that is have every signal change to green as you approach it. One day you notice as you come over the brow of a hill that every traffic light you can see has just changed to green and that therefore your chances of catching every signal is slight. As you wait at a red light you begin to wonder how long it will be before all the lights again show green, not necessarily all turn green, merely all show green simultaneously, even if it is only for a second.

Write a program that will determine whether this event occurs within a reasonable time. Time is measured from the instant when they all turned green simultaneously, although the initial portion while they are all still green is excluded from the reckoning.

Input

Input will consist of a series of scenarios. Data for each scenario will consist of a series of integers representing the cycle times of the traffic lights, possibly spread over many lines, with no line being longer than 100 characters. Each number represents the cycle time of a single signal. The cycle time is the time that traffic may move in one direction; note that the last 5 seconds of a green cycle is actually orange. Thus the number 25 means a signal that (for a particular direction) will spend 20 seconds green, 5 seconds orange and 25 seconds red. Cycle times will not be less than 10 seconds, nor more than 90 seconds. There will always be at least two signals in a scenario and never more than 100. Each scenario will be terminated by a zero (0). The file will be terminated by a line consisting of three zeroes (0 0 0).

Output

Output will consist of a series of lines, one for each scenario in the input. Each line will consist of the time in hours, minutes and seconds that it takes for all the signals to show green again after at least one of them changes to orange. Follow the format shown in the examples. Time is measured from the instant they all turn green simultaneously. If it takes more than five hours before they all show green simultaneously, the message "`Signals fail to synchronise in 5 hours`" should be written instead.

Sample input

```
19 20   0
30
25 35 0
0 0 0
```

Sample output

```
00:00:40
00:05:00
```

Will Indiana Jones Get There?

Program indiana.c, indiana.cpp, indiana.java, indiana.pas

Indiana Jones is in a deserted city, annihilated during a war. Roofs of all houses have been destroyed and only portions of walls are still standing. The ground is so full of mines that the only safe way to move around the city is walking over the remaining walls. The mission of our hero is to save a person who is trapped in the city. In order to move between two walls which are not connected Indiana Jones thought of taking with him a wooden board which he could place between the two walls and then cross from one to the other.

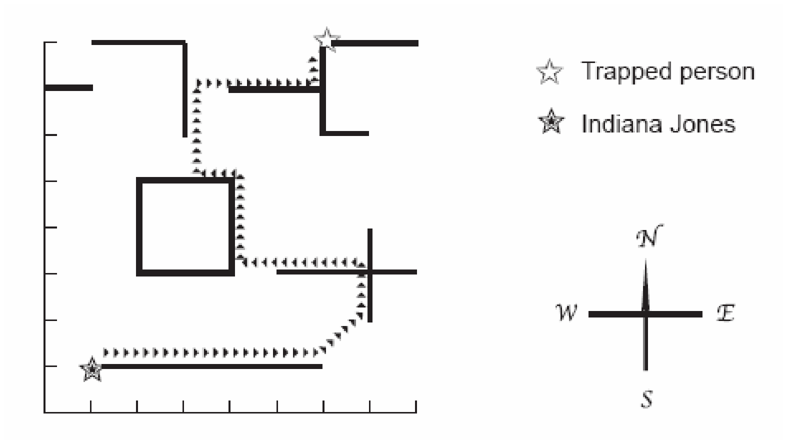


Fig. 1: City map with route used by Indiana Jones

Initial positions of Indiana Jones and the trapped person are both on some section of the walls. Besides, walls are either in the direction South-North or West-East. You will be given a map of the city remains. Your mission is to determine the minimum length of the wooden board Indiana Jones needs to carry in order to get to the trapped person.

Input

Your program should process several test cases. Each test case starts with an integer *N* indicating the number of wall sections remaining in the city ($2 \le N \le 1000$). Each of the next *N* lines describes a wall section. The first wall section to appear is the section where Indiana Jones stands at the beginning. The second section to appear is the section where the trapped person stands. Each wall section description consists of three integers *X*, *Y* and *L* ($-10000 \le X, Y, L \le 10000$), where *X* and *Y* define either the southernmost point of a wall section (for South-North sections) or the westernmost point (for West-East wall sections). The value of *L* determines the length and direction of the wall: if $L \ge 0$, the section is West-East, with length *L*; if $L < 0$, the section is North-South, with length $|L|$. The end of input is indicated by $N = 0$.

Output

For each test case in the input your program should produce one line of output, containing a real value representing the length of the wooden board Indiana Jones must carry. The length must be printed as a real number with two-digit precision. The input will not contain test cases where differences in rounding are significant.

Sample input

```
14
1 1 5
6 8 2
7 2 -2
5 3 3
2 5 2
2 3 2
2 3 -2
4 3 -2
0 7 1
1 8 2
3 6 -2
4 7 2
6 6 1
6 6 -2
3
-10 0 20
-5 1 10
50 50 100
0
```

Sample output

```
1.41
1.00
```

Simply Syntax

Program syntax.c, syntax.cpp, syntax.java, syntax.pas

In the land of Hedonia the official language is Hedonian. A Hedonian professor had noticed that many of her students still did not master the syntax of Hedonian well. Tired of correcting the many syntactical mistakes, she decided to challenge the students and asked them to write a program that could check the syntactical correctness of any sentence they wrote. Similar to the nature of Hedonians, the syntax of Hedonian is also pleasantly simple. Here are the rules:

The only characters in the language are the characters p through z and N, C, D, E, and I.

Every character from p through z is a correct sentence.

If *s* is a correct sentence, then so is N*s*.

If *s* and *t* are correct sentences, then so are C*st*, D*st*, E*st*, and I*st*.

Rules 0. to 3. are the only rules to determine the syntactical correctness of a sentence.

You are asked to write a program that checks if sentences satisfy the syntax rules given in Rule 0. - Rule 4.

Input:

The input consists of a number of sentences (1 or more) consisting only of characters p through z and N, C, D, E, and I. Each sentence is ended by a new-line character. The collection of sentences is terminated by the end-of-file character. If necessary, you may assume that each sentence has at most 256 characters and at least 1 character.

Output:

The output consists of the answers YES for each well-formed sentence and NO for each not-well-formed sentence. The answers are given in the same order as the sentences. Each answer is followed by a new-line character, and the list of answers is followed by an end-of-file character.

Sample Input:

```
Cp
Isz
NIsz
Cqpq
```

Sample Output:

```
NO
YES
YES
NO
```

Student Grants [Kci naloga]

Program grants.c, grants.cpp, grants.java, grants.pas

The Government of Impecunia has decided to discourage tertiary students by making the payments of tertiary grants a long and time-consuming process. Each student is issued a student ID card which has a magnetically encoded strip on the back which records the payment of the student grant. This is initially set to zero. The grant has been set at $40 per year and is paid to the student on the working day nearest to his birthday. (Impecunian society is still somewhat medieval and only males continue with tertiary education.) Thus on any given working day up to 25 students will appear at the nearest office of the Department of Student Subsidies to collect their grant.

The grant is paid by an Automatic Teller Machine which is driven by a reprogrammed 8085½ chip originally designed to run the state slot machine. The ATM was built in the State Workshops and is designed to be difficult to rob. It consists of an interior vault where it holds a large stock of $1 coins and an output store from which these coins are dispensed. To limit possible losses it will only move coins from the vault to the output store when that is empty. When the machine is switched on in the morning, with an empty output store, it immediately moves 1 coin into the output store. When that has been dispensed it will then move 2 coins, then 3, and so on until it reaches some preset limit *k*. It then recycles back to 1, then 2 and so on.

The students form a queue at this machine and, in turn, each student inserts his card. The machine dispenses what it has in its output store and updates the amount paid to that student by writing the new total on the card. If the student has not received his full grant, he removes his card and rejoins the queue at the end. If the amount in the store plus what the student has already received comes to more than $40, the machine only pays out enough to make the total up to $40. Since this fact is recorded on the card, it is pointless for the student to continue queuing and he leaves. The amount remaining in the store is then available for the next student.

Write a program that will read in values of *N* (the number of students, $1 \le N \le 25$) and *k* (the limit for that machine, $1 \le k \le 40$) and calculate the order in which the students leave the queue.

Input and Output

Input will consist of a series of lines each containing a value for *N* and *k* as integers. The list will be terminated by two zeroes (0 0).

Output will consist of a line for each line of input and will contain the list of students in the order in which they leave the queue. Students are ordered according to their position in the queue at the start of the day. All numbers must be right justified in a field of width 3.

Sample input

```
5 3
0 0
```

Sample output

```
  1  3  5  2  4
```