Craiova, județul Dolj

Concursul de informatică

ȘTEFAN ODOBLEJA

În perioada 5-7 martie 2002 s-a desfășurat la Craiova un concurs interjudețean de programare la care au participat elevi din nouă județe ale țării. În continuare vă prezentăm enunțurile celor zece probleme cu care s-au confruntat concurenții.

Clasa a IX-a

P050225: Puzzle

(lect. univ. dr. Petre Băzăvan)

Se consideră o imagine alcătuită din mai multe piese de formă dreptunghiulară, fiecare având aceleași dimensiuni. Imaginea este formată din n linii și n coloane, fiecare element al acesteia fiind o piesă. Elementele componente ale unei piese sunt simboluri din setul de caractere *ASCII*. Sarcina voastră este de a reconstitui imaginea inițială pe baza pieselor componente. Fiecare piesă este caracterizată prin patru numere, corespunzătoare celor patru laturi ale sale. Dacă unul dintre aceste numere este 0, înseamnă că latura respectivă se află pe marginea imaginii inițiale. Dacă valoarea unui număr este x ($x \neq 0$), atunci latura respectivă se va învecina cu latura unei alte piese căreia îi corespunde numărul $-x$. Piesele nu pot fi rotite și sunt unic determinate de cele patru numere care descriu marginile.

Date de intrare

Fișierul de intrare **PUZZLE.IN** are următoarea structură:

- prima linie conține trei numere naturale n, h, l, unde n reprezintă dimensiunea imaginii (numărul de piese) care trebuie reconstituită, iar numerele h și l reprezintă înălțimea, respectiv lățimea unei piese;
- următoarele linii conțin descrierea celor n^2 piese; acestea vor fi separate prin câte o linie vidă; fiecare piesă va fi descrisă prin $h + 1$ linii astfel:
  - ♦ primele h linii vor conține câte l caractere care descriu liniile piesei;
  - ♦ următoarea linie va conține cele patru numere care descriu marginile piesei; numerele corespunzătoare laturilor apar în ordinea *sus*, *stânga*, *jos*, *dreapta*.

Date de ieșire

Fișierul de ieșire **PUZZLE.OUT** va conține $n \cdot h$ linii, pe fiecare dintre acestea aflându-se $n \cdot l$ caractere. Aceste caractere vor descrie imaginea reconstituită.

Restricții și precizări

- $2 \leq n \leq 10$;
- $1 \leq h, l \leq 25$;
- laturile pieselor sunt caracterizate prin numere întregi cuprinse între -5 și 5;
- întotdeauna există o singură soluție;
- spațiile sunt caractere valide care pot intra în componența pieselor.

Exemplu

PUZZLE.IN

```
2 8 14
88,
8888.
:8888b
  8888
-.:888b
' d8888
 ,88888
':88888
0 3 -1 0

          o8%88
       o88%888
      8'-    -
     8'
    d8.-=. ,==
    >8 `~` :`~
    88
    88b. `-~
0 0 -5 -3

.:88888
:::8888
:' 8888b
    8888b
    ,%888b.
    %%%8--'-.
   _%-' ---  -
.-'    =  --.
1 4 0 0

    888b ~==~
    88888o--:'
    `88888| ::
    8888^^'
   d888
  d88%
 /88:.__ ,
     '''::===.
5 0 0 -4
```

PUZZLE.OUT

```
          o8%8888,
        o88%8888888.
       8'-    -:8888b
      8'          8888
     d8.-=. ,==-.:888b
     >8 `~` :`~' d8888
     88          ,88888
     88b. `-~  ':88888
     888b ~==~ .:88888
     88888o--:':::8888
     `88888| :::' 8888b
     8888^^'       8888b
    d888            ,%888b.
   d88%             %%%8--'-.
  /88:.__ ,       _%-' ---  -
      '''::===..-'    =  --.
```

Timp de execuție: 1 secundă/test

Despre concurs...

Concursul de programare "Ștefan Odobleja" a fost organizat de Liceul de Informatică din Craiova, în colaborare cu Inspectoratul Școlar Județean Gorj și Facultatea de Matematică -Informatică din cadrul Universității din Craiova.

La concurs au participat 44 de elevi din nouă județe ale țării: Alba, Brașov, Cluj, Dolj, Gorj, Iași, Mehedinți, Olt și Vâlcea.

Au fost două secțiuni: una la care concurenții au participat individual și una la care reprezentanții fiecărui județ, indiferent de clasă, au format echipe.

Redacția GInfo a fost reprezentată la acest concurs de redactorul șef Clara Ionescu, precum și de redactorul Eugen Ionescu.

P040226: Ecuații

(asist. univ. Florin Stavu)

Scrieți un program care rezolvă ecuații de gradul întâi. Ecuațiile sunt descrise potrivit următoarei sintaxe:

- *ecuație*: *expresie* `=` *expresie*;
- *expresie*: *termen* **sau** *expresie* (`+` **sau** `-`) *termen*;
- *termen*: *factor* **sau** *factor* `*` *factor*;
- *factor*: *număr* **sau** `x` **sau** (*expresie*);
- *număr*: *cifră* **sau** *cifră număr*;
- *cifră*: `0` **sau** `1` **sau** `...` **sau** `9`.

Conform acestei sintaxe nu se pot scrie ecuații care să conțină termeni de forma `x*x`. Se garantează că fișierul de intrare va conține doar ecuații liniare în *x* (ecuații de gradul întâi).

De asemenea, toate subexpresiile unei ecuații vor fi liniare în *x* (de exemplu, nu poate apărea expresia `x*x-x*x+3*x=9`).

Date de intrare

Fișierul de intrare **`ECUATII.IN`** conține mai multe ecuații, câte una pe fiecare linie.

Date de ieșire

În fișierul de ieșire **`ECUATII.OUT`** se vor scrie rezultatele ecuațiilor. Pentru fiecare ecuație se vor scrie două linii: prima dintre ele va conține textul `Ecuatia #i`, unde `i` este numărul de ordine al ecuației; a doua linie va conține unul dintre următoarele mesaje:

- `Nu are solutie`, în cazul în care ecuația nu are soluții;
- `O infinitate de soluții`, în cazul în care ecuația are o infinitate de soluții;
- `x=valoare`, unde valoare este numărul real (tipărit cu șase cifre zecimale) care reprezintă valoarea soluției, în cazul în care există o singură soluție a ecuației.

Restricție

- fiecare ecuație este descrisă prin cel mult 100 de caractere.

Exemplu

`ECUATII.IN`	**`ECUATII.OUT`**
`x+x=10`	`Ecuatia #1`
`3*x+2=21`	`x = 5.000000`
`3*x=3*x+1+2+3`	`Ecuatia #2`
`(12-3*4)*x=2*5-10`	`x = 6.333333`
	`Ecuatia #3`
	`Nu are solutie`
	`Ecuatia #4`
	`O infinitate de solutii`

Timp de execuție: 1 secundă/test

Clasa a X-a

P040227: Arheolog

(asist. univ. Mihai Stancu)

Un arheolog care cerceta vestigiile unei civilizații dispărute cu mii de ani în urmă, a găsit șapte plăcuțe hexagonale pe care erau desenate diverse inscripții. Dorind să descifreze mesajul, el descoperă că plăcuțele trebuie alipite într-un anume fel. Arheologul așează cele șapte plăcuțe (identificate prin literele `A`, `B`, `C`, `D`, `E`, `F` și `G`) pe șapte tije verticale (identificate prin numerele `0`, `1`, `2`, `3`, `4`, `5` și `6`; tija din centru este identificată prin `0`). În figura 1 este ilustrat modul de amplasare a tijelor.

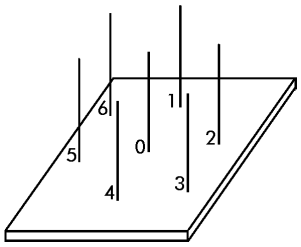


Fig. 1: Tijele

Două plăcuțe hexagonale se pot îmbina dacă au același tip de striuri pe laturile care vor fi învecinate. Există exact zece tipuri de striuri, identificate prin cele zece cifre. În figura 2 este ilustrată plăcuța `D` care are laturi cu diferite tipuri de striuri.

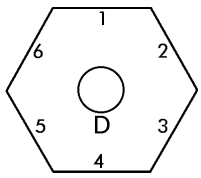


Fig. 2: Plăcuță

Cele șapte hexagoane sunt amplasate pe cele șapte tije, fiecare hexagon din margine având trei laturi lipite de laturile altor hexagoane, în timp ce hexagonul de pe tija `0` are toate laturile lipite de cele ale altor hexagoane. Pentru a ajuta la rezolvarea misterului, va trebui să găsiți un aranjament și o orientare pentru plăcuțele hexagonale, astfel încât striurile de pe oricare două laturi lipite să fie identice. O configurație corectă este ilustrată în figura 3.

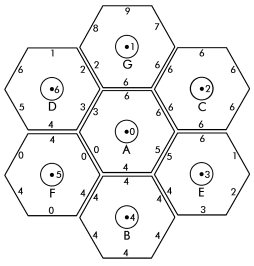


Fig. 3: Amplasament corect

Date de intrare

Fișierul **ARHEOLOG.IN** conține mai multe seturi de date corespunzătoare mai multor descrieri ale plăcuțelor. Fiecare set de plăcuțe va fi descris prin opt linii. Primele șapte linii conțin câte șase cifre (neseparate prin spații) care indică tipurile de striuri de pe laturile plăcuțelor. Prima linie conține cifrele corespunzătoare hexagonului `A`, a doua linie conține cifrele corespunzătoare hexagonul `B` etc. Prima cifră de pe fiecare linie corespunde laturii de sus, iar următoarele corespund celorlalte laturi (parcurse în sensul acelor de ceasornic). De exemplu, hexagonul din figura 1 este reprezentat prin secvența de cifre `123456`. Cea de-a opta linie va conține doar caracterul `*` și are rolul de a delimita seturile de date.

Date de ieșire

Fișierul de ieșire **ARHEOLOG.OUT** va conține câte un tabel pentru fiecare set de date. Acesta va fi format dintr-un cap de tabel și șapte linii a câte trei coloane. Prima coloană corespunde tijelor, cea de-a doua hexagoanelor, iar cea de-a treia orientării plăcuțelor. În prima coloană se vor scrie numerele de identificare a tijelor. Pe cea de-a doua coloană se va scrie litera care identifică plăcuța care va fi amplasată pe tija din prima coloană. Cea de-a treia coloană va conține tipurile de striuri de pe cele șase laturi ale hexagonului; prima cifră va corespunde laturii de sus, iar următoarele corespund celorlalte laturi (parcurse în sensul acelor de ceasornic). Capul de tabel va conține textul `TIJ` pentru prima coloană, `HEX` pentru a doua și `POZITIE` pentru a treia.

În cazul în care nu există nici o soluție, se va scrie doar mesajul `Nu exista solutii`.

Datele de ieșire corespunzătoare unui set de date vor fi urmate de o linie care va conține 20 de caractere `*`.

Exemplu

ARHEOLOG.IN
```
665403
444444
666666
123456
123456
040404
289766
*
123456
111111
222222
333333
444444
555555
666666
*
```

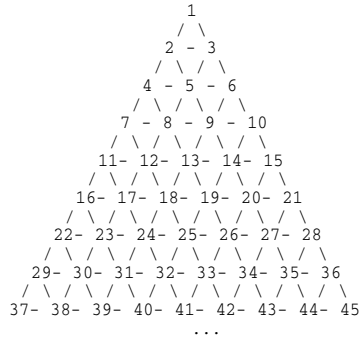
ARHEOLOG.OUT
```
TIJ    HEX    POZITIE
 0      A     665403
 1      G     976628
 2      C     666666
 3      E     612345
 4      B     444444
 5      F     404040
 6      D     123456
********************
Nu exista solutii.
********************
```

Timp de execuție: 1 secundă/test

P030228: Numere

(asist. univ. Mihai Gabroveanu)

Se consideră o grilă infinită de puncte, reprezentând vârfuri ale unor triunghiuri echilaterale așezate astfel:



Aceste puncte sunt numerotate de la stânga la dreapta și de sus în jos. Grupuri de puncte pot forma laturile unor figuri geometrice. De exemplu, mulțimile de puncte `{1,2,3}` și `{7,9,18}` reprezintă vârfurile unor triunghiuri, mulțimile `{11,13,26,24}` și `{2,7,9,18}` reprezintă vârfurile unor paralelograme, iar mulțimile `{4,5,9,13,12,7}` și `{8,10,17,21,32,34}` reprezintă vârfurile unor hexagoane.

Va trebui să scrieți un program care stabilește dacă o mulțime de puncte de pe această grilă reprezintă laturile unei figuri acceptate: *triunghi*, *paralelogram*, *hexagon*.

O figură acceptată trebuie să îndeplinească următoarele condiții:

- fiecare latură a figurii trebuie să coincidă cu o muchie din grilă (prin muchie se înțelege unul, două sau mai multe segmente care unesc puncte adiacente coliniare);
- laturile figurii trebuie să aibă lungimi egale.

Date de intrare

Intrarea va consta dintr-un număr necunoscut de mulțimi de puncte. Fiecare mulțime de puncte va apărea pe câte o linie a fișierului **NUMERE.IN** . Fiecare mulțime conține cel mult șase puncte, iar numerele corespunzătoare acestora sunt cuprinse între 1 și 32767.

Date de ieșire

Fișierul de ieșire **NUMERE.OUT** va conține câte o linie pentru fiecare mulțime din fișierul de intrare. Pe această linie se va scrie un mesaj care va descrie tipul figurii reprezentate de punctele mulțimii: `TRIUNGHI`, `PARALELOGRAM` sau `HEXAGON`.

În cazul în care figura corespunzătoare nu este acceptată, linia va conține mesajul `NU`.

Exemplu

NUMERE.IN	NUMERE.OUT
1 2 3	TRIUNGHI
11 13 29 31	NU
26 11 13 24	PARALELOGRAM
4 5 9 13 12 7	HEXAGON
1 2 3 4 5	NU
47	NU
11 13 23 25	NU

Timp de execuție: 1 secundă/test

Clasele a XI-a și a XII-a

P030229: Partiție

(asist. univ. Mirel Coșulschi, asist. univ. Nicolae Constantinescu)

Fie un număr natural M; numim partiție a lui M două secvențe de numere întregi $x_1, x_2, ..., x_k$ și $n_1, n_2, ..., n_k$, astfel încât:

- $x_1 < x_2 < ... < x_k$;
- $n_1 \cdot x_1 + n_2 \cdot x_2 + ... + n_k \cdot x_k = M$.

Prin sumă parțială a unei partiții se înțelege expresia $m_1 \cdot x_1 + m_2 \cdot x_2 + ... + m_k \cdot x_k$, unde $m_i \le n_i$.

Se cere să se determine o partiție a numărului M care indeplinește următoarele condiții:

- orice număr cuprins între 1 și M poate fi reprezentată în mod unic ca o sumă parțială a acestei partiții;
- partiția conține un număr maxim de elemente distincte.

De exemplu, pentru $M = 7$, câteva dintre posibilitățile de partiționare sunt:

- $7 = 7 \cdot 1$ ($x_1 = 1, n_1 = 7$);
- $7 = 1 \cdot 1 + 3 \cdot 2$ ($x_1 = 1, x_2 = 2, n_1 = 1, n_2 = 3$);
- $7 = 1 \cdot 1 + 1 \cdot 2 + 1 \cdot 4$ ($x_1 = 1, x_2 = 2, x_3 = 4, n_1 = 1, n_2 = 1, n_3 = 1$);
- $7 = 3 \cdot 1 + 2 \cdot 2$ ($x_1 = 1, x_2 = 2, n_1 = 3, n_2 = 2$).

Ultima dintre aceste partiții nu îndeplinește prima condiție, deoarece numărul 2 poate fi scris în două moduri ca sumă parțială a partiției:

- $2 = 2 \cdot 1 + 0 \cdot 2$ ($m_1 = 2, m_2 = 0$);
- $2 = 0 \cdot 1 + 1 \cdot 2$ ($m_1 = 0, m_2 = 1$).

Celelalte partiții îndeplinesc această condiție; ca urmare, partiția căutată este $1 \cdot 1 + 1 \cdot 2 + 1 \cdot 4$, deoarece conține mai multe elemente decât celelalte.

Date de intrare

Fișierul de intrare **PARTITIE.IN** conține o singură linie pe care se află valoarea M.

Date de ieșire

Fișierul de ieșire **PARTITIE.OUT** va conține trei linii. Pe prima dintre ele se va afla valoarea k care indică numărul de elemente ale partiției.

Cea de-a doua linie va conține valorile $n_1, n_2, ..., n_k$, separate prin câte un spațiu.

Ultima linie va conține valorile $x_1, x_2, ..., x_k$, separate prin câte un spațiu.

Restricție

- $1 \le M \le 255$;

Exemplu

PARTITIE.IN	PARTITIE.OUT
7	3 1 1 1 1 2 4

Timp de execuție: 1 secundă/test

P030230: Bile

(asist. univ. Mirel Coșulschi, asist. univ. Nicolae Constantinescu)

Un distribuitor de bile este un aparat cu o intrare și două ieșiri (*stângă* și *dreaptă*) care funcționează astfel: la intrare primește un șir de bile pe care le distribuie alternativ prin cele două ieșiri: prima bilă iese prin stânga, a doua prin dreapta, a treia prin stânga etc.

Să considerăm acum un arbore format din astfel de distribuitoare. Într-un astfel de arbore bilele se introduc prin vârf și trec succesiv prin distribuitoare în funcție de starea fiecărui distribuitor, până ajung la o ieșire care nu este conectată la nici un distribuitor; spunem că în acest moment bila *a părăsit arborele*.

Dacă ieșirea prin care bila părăsește arborele este ieșirea din stânga a unui distribuitor, atunci se spune că bila *a părăsit arborele prin stânga*; analog, dacă ieșirea prin care bila părăsește arborele este ieșirea din dreapta a unui distribuitor, atunci se spune că bila *a părăsit arborele prin dreapta*.

Vom considera un aparat format dintr-un astfel de arbore sub care se află o tăviță cu N bile, numerotate de la 1 la N, așezate inițial în ordine crescătoare de la stânga la dreapta.

La fiecare pas toate bilele sunt introduse în arbore, în ordinea în care se aflau în tăviță, de la stânga la dreapta. Fiecare bilă care va părăsi arborele prin stânga se va așeza în stânga bilelor prezente în tăviță în momentul căderii ei și, analog, fiecare bilă care va părăsi arborele prin dreapta se va așeza în dreapta bilelor din tăviță.

O bilă este introdusă în arbore doar după ce predecesoarea sa a ajuns deja în tăviță. După fiecare astfel de pas, toate bilele se vor afla în tăviță, dar în altă ordine.

Pentru o configurație dată, să se determine numărul minim de pași după care bilele ajung în tăviță în ordinea inițială.

Date de intrare

Fișierul de intrare **BILE.IN** are următoarea structură:

- pe prima linie se află două numere întregi pozitive N și M care reprezintă numărul bilelor, respectiv cel al distribuitoarelor;
- pe următoarele M linii se află câte trei numere întregi pozitive separate printr-un singur spațiu i, j, k care indică faptul că distribuitorul j este conectat la ieșirea stângă a distribuitorului i, iar distribuitorul k este conectat la ieșirea dreaptă a distribuitorului i; în cazul în care una dintre valorile j sau k este 0, atunci ieșirea corespunzătoare nu este conectată la nici un distribuitor.

Date de ieșire

Fișierul de ieșire **BILE.OUT** va conține o singură linie pe care se va afla numărul minim de pași necesari pentru ca bilele să se afle în tăviță în ordinea inițială.

Restricții și precizări

- $0 < N, M \le 1000$;

- în vârf se află întotdeauna distribuitorul cu numărul de ordine 1;
- după fiecare pas distribuitoarele sunt "resetate", astfel încât prima bila pe care o vor primi la pasul următor o vor distribui prin stânga.

Exemplu

BILE.IN	BILE.OUT
4 4 1 2 3 2 0 0 3 4 0 4 0 0	2

Timp de execuție: 1 secundă/test

P030231: For

(asist. univ. Mirel Coșulschi)

Fie următorul fragment de program scris în limbajul *Pascal*:

```
...
for i:=1 to 20 do
  for j := 1 to i do
    write('*');
...
```

Se poate verifica foarte ușor faptul că instrucțiunea formată din apelul procedurii `write` se va executa de 210 ori.

Se dau mai multe instrucțiuni de ciclare **`for`** imbricate, corpul ultimei instrucțiuni de ciclare conținând un apel `write`.

Se cere să se scrie un program care să determine de câte ori se va apela procedura `write` în timpul execuției fragmentului respectiv.

Date de intrare

Fișierul de intrare **`FOR.IN`** va conține pe prima linie numărul *n* al instrucțiunilor de ciclare imbricate. Pe fiecare dintre următoarele *n* linii se va afla un triplet de forma *var*, exp_1, exp_2; *var* este un identificator valid în limbajul *Pascal* și reprezintă variabila de ciclare pentru o instrucțiune **`for`**; exp_1 și exp_2 pot fi constante sau identificatorul variabilei de ciclare corespunzatoare instrucțiunii **`for`** de pe nivelul anterior.

Date de ieșire

Fișierul de ieșire **`FOR.OUT`** va conține o singură linie pe care se va afla numărul de execuții al instrucțiunii care conține apelul procedurii `write`.

Restricții și precizări

- $0 < n < 20$;
- constantele prezente în fișierul de intrare sunt numere naturale mai mici decât 1000.

Exemplu

FOR.IN	FOR.OUT
2 i 1 20 j 1 i	210

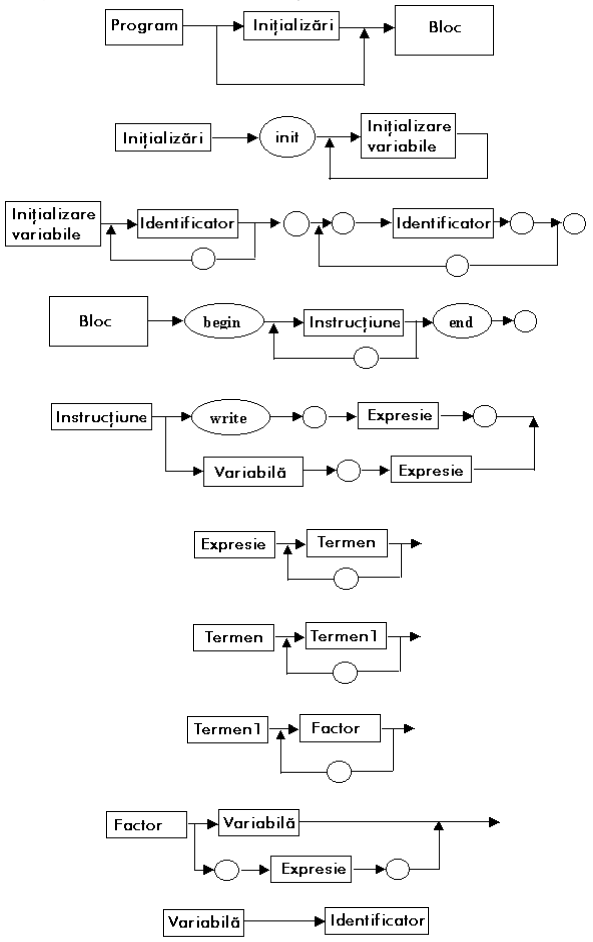
Timp de execuție: 1 secundă/test

Concursul pe echipe

P030232: MultiX

(asist. univ. Mirel Coșulschi)

Cosiderăm un nou limbaj de programare pe care îl definim cu ajutorul următoarelor diagrame de sintaxă:



Identificatorii sunt definiți într-un mod asemănător cu cel din *Pascal*: orice combinație de litere și cifre, care începe obligatoriu cu o literă și are lungimea mai mică decât 255 de caractere. O variabilă identifică o mulțime.

Se definesc următoarele operații:

- **`*`** - intersecția a două mulțimi;
- **`+`** - reuniunea a două mulțimi;
- **`*`** - diferența a două mulțimi;

Numim acest limbaj *MultiX*. Fie următorul program scris în acest limbaj:

```
init a : "a", "b", "c";
     b,c : "a", "c", "d";
```

```
begin
  a := a + b;
  write(a);
  b := a - (a * b);
  write(b);
end.
```

Interpretarea acestui program este următoarea:

- se inițializează mulțimile A = {'a', 'b', 'c'}, B = {'a', 'c', 'd'} și C = {'a', 'c', 'd'};
- prin instrucțiunea `a := a + b` se calculează reuniunea mulțimilor A și B, iar rezultatul este atribuit variabilei care reprezintă mulțimea A; această mulțime devine A = {'a', 'b', 'c', 'd'};
- prin instrucțiunea `write(a)` se afișează elementele mulțimii A; așadar, se va afișa `a b c d`;
- prin instrucțiunea `b := a - (a * b)` se calculează diferența dintre mulțimea A și intersecția mulțimilor A și B, iar rezultatul este atribuit variabilei care reprezintă mulțimea B; această mulțime devine B = {'b'};
- prin instrucțiunea `write(b)` se afișează elementele mulțimii B; așadar, se va afișa `b`.

Să se realizeze o aplicație care, pentru un program scris în limbajul *MultiX*, îi validează corectitudinea conform diagramelor de sintaxă anterioare și, în cazul unui program valid, va afișa rezultatul execuției acestuia.

Date de intrare

Fișierul de intrare **`MULTIX.IN`** va conține un program scris în limbajul *MultiX*.

Date de ieșire

Dacă programul conține erori sintactice, fișierul de ieșire **`MULTIX.OUT`** va conține mesajul `Eroare sintactica pe linia x`, unde `x` este numărul liniei pe care a apărut prima eroare. În cazul în care nu există erori sintactice, fișierul va conține rezultatele afișate de program. Fiecare linie a fișierului va corespunde unui apel `write`; elementele mulțimilor vor fi separate prin câte un spațiu.

Restricții și precizări

- numărul variabilelor prezente în program este de cel mult 255;
- cardinalul unei mulțimi nu va depăși valoarea 1000;
- fiecare linie a programului conține o singură instrucțiune;
- o variabilă care nu apare în secțiunea de inițializare se consideră a fi inițializată cu mulțimea vidă;
- limbajul *MultiX* este *case sensitive*: se fac deosebiri între litere mici și litere mari;
- un apel de scriere, al cărui argument este mulțimea vidă, va duce la tipărirea simbolului `$` pe linia corespunzătoare;
- o linie a programului poate conține cel mult 1000 de caractere.

Exemplu

`MULTIX.IN`

```
init x: "a1","a2","a3";
     y,z: "u","a2","z";
begin
  write(x * y);
  z := z + (y - x);
  write(z);
end.
```

`MULTIX.OUT`

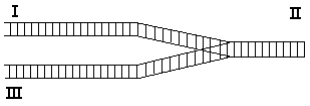
```
a2
u a2 z
```

Timp de execuție: 1 secundă/test

P030233: Depou

(asist. univ. Mirel Coșulschi)

Într-un depou cu trei linii, suficient de lungi, avem o garnitură de tren formată din N vagoane, numerotate de la 1 la N. Inițial vagoanele se află pe linia I și garnitura trebuie să ajungă pe linia III.



Un vagon se poate deplasa de pe linia I pe linia II (dacă nu există nici un vagon la dreapta sa) sau de pe linia II pe linia III (dacă nu există nici un vagon la stânga sa). Vagoanele nu se pot deplasa în sens invers (de pe linia II pe linia I sau de pe linia III pe linia II) sau direct de pe linia I pe linia III. La fiecare pas se poate deplasa de pe o linie pe alta un singur vagon. Să considerăm trenul format din vagoanele 1, 2 și 3.

Premianții - concursul individual

Clasa a IX-a

Andrei Ioniță, Olt - premiul I
Tudor-Alexandru Lițu, Iași - premiul II
Alexandru Crețu, Dolj - premiul III
Ciprian Stănescu, Dolj - premiul III

Clasa a X-a

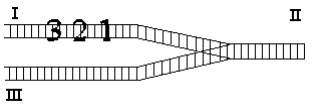
Flaviu Roman, Cluj - premiul I
Radu Pârvu, Brașov - premiul II
Iolanda Popa, Iași - premiul III

Clasa a XI-a

Alexandru Surpățean, Brașov - premiul I
Horia-Andrei Ciochină, Iași - premiul II
Cezar Stalea, Dolj - premiul III
Marius Truică, Dolj - premiul III

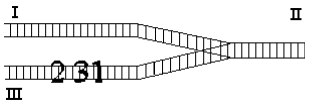
Clasa a XII-a

Bogdan Hârjoc, Alba - premiul I
Mihai Iacob, Brașov - premiul II
Cristian Ivănescu, Dolj - premiul III



O posibilitate de mutare a vagoanelor de pe linia I pe linia III ar putea fi:
- pasul 1: vagonul 1 trece de pe linia I pe linia II;
- pasul 2: vagonul 2 trece de pe linia I pe linia II;
- pasul 3: vagonul 2 trece de pe linia II pe linia III;
- pasul 4: vagonul 3 trece de pe linia I pe linia II;
- pasul 5: vagonul 3 trece de pe linia II pe linia III;
- pasul 6: vagonul 1 trece de pe linia II pe linia III;

După efectuarea acestor pași, vagoanele ajung pe linia III în ordinea 2, 3, 1.



Să se scrie un program care determină numărul variantelor în care pot fi așezate vagoanele pe linia III.

Date de intrare
Fișierul de intrare **DEPOU.IN** va conține numărul vagoanelor de pe linia I a depoului.

Date de ieșire
Fișierul de ieșire **DEPOU.OUT** va conține numărul configurațiilor posibile în care se pot afla vagoanele pe linia III a depoului.

Restricție
- $1 \le N \le 100$.

Exemplu

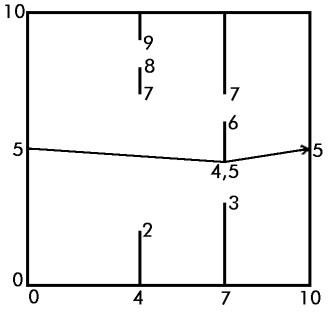
DEPOU.IN	DEPOU.OUT
3	5

Timp de execuție: 1 secundă/test

P030234: Cameră

(asist. univ. Mihai Gabroveanu)

Să se determine lungimea celui mai scurt drum printr-o cameră ce conține pereți despărțitori. Camera va avea întotdeauna ca margini pereții delimitați de punctele de coordonate (0, 0), (10, 0), (10, 10), (0, 10), după cum se poate vedea în figura de mai jos. Punctul de plecare este întotdeauna (0, 5), iar punctul de sosire este (10, 5). În cameră pot exista k pereți despărțitori, fiecare dintre ei având două uși. Figura prezintă o cameră cu doi pereți despărțitori; este ilustrat și drumul minim corespunzător camerei respective.



Date de intrare
Fișierul de intrare **CAMERA.IN** conține pe prima linie numărul k al pereților despărțitori. Fiecare dintre următoarele k linii conține descrierea unui perete. O linie va conține cinci numere reale; primul dintre ele reprezintă coordonata orizontală a peretelui. Următoarele două numere reprezintă coordonatele verticale ale primei uși, în timp ce ultimele două corespund coordonatelor verticale ale celei de a doua uși.

Date de ieșire
Fișierul de ieșire **CAMERA.OUT** va conține o singură linie pe care se va afla un număr real (scris cu două zecimale exacte) care reprezintă lungimea celui mai scurt drum.

Restricție
- $0 \le k \le 18$.

Exemplu

CAMERA.IN	CAMERA.OUT
2 4 2 7 8 9 7 3 4.5 6 7	10.06

Timp de execuție: 1 secundă/test

Premianții - concursul pe echipe

Premiul I
Andrei Homescu, Emanuel Udrea, Silviu Udrea - Gorj

Premiul II
Cristian Domșa, Bogdan Hârjoc, Ionuț Turus - Alba

Premiul III
Mircea Ispas, Constantin Jianu, Lucian Morăreț, Mihai Udrea - Dolj