```
{ Se considera o tabla de sah de dimensiune 8 x 8, care are
  anumite casute perforate (din totalul de 64 de casute unele
  sunt perforate). Lista de casute perforate se precizeaza prin
  coordonatele lor. Pentru fiecare casuta perforata se specifica
  linia si coloana ei. Liniile si coloanele se numeroteaza de la
  1 pana la 8.

  In coltul de coordonate (1, 1) al tablei se afla un cal (piesa
  de sah care se deplaseaza prin sarituri in forma de L).

  Scopul este sa gasim un drum pe care calul sa ajunga in coltul
  de coordonate (8, 8) al tablei de sah.

  Nu are importanta lungimea drumului (numarul de sarituri efectuate
  de cal). Adica nu se cere sa se gaseasca drumul cel mai scurt. Avem
  o singura restrictie: calul nu are voie sa treaca prin casutele
  perforate.

  Casutele (1, 1) si (8, 8) nu sunt perforate.
}
program Drumul_calului;

uses Crt;

{ Definim dimensiunea tablei de sah ca si constanta. }
const N = 8;

type
    { Tabla de sah o vom pastra sub forma unei matrici de dimensiuni
      N x N. Daca o casuta de pe tabla nu este perforata, atunci in
      matrice vom pastra 0 pe pozitia corespunzatoare casutei respective.
      Daca o casuta de pe tabla este perforata, atunci in matrice vom
      pastra 1 pe pozitia corespunzatoare casutei respective. }
    TTabla = array[1..N, 1..N] of Integer;

    { Lista de casute gaurite presupune pastrarea unor perechi de coordonate
      (linie, coloana). Definim un tip de date care pastreaza o lista
      de asemenea perechie de coordonate. }
    TListaCoordonate = array[1..N*N, 1..2] of Integer;

var
   { Definim o variabila care pastreaza configuratia tablei de sah. }
   tabla : TTabla;

   { Apoi definim o variabila care pastreaza coordonatele casutelor
     perforate. }
   perforatii : TListaCoordonate;
   { Si inca o variabila care pastreaza numarul casutelor perforate. }
   nr_perf : Integer;

   { Drumul calului pe tabla de sah se poate pastra tot sub forma unei
     liste de coordonate. Pastram coordonatele tuturor casutelor pe unde
     trece calul pe tabla. Definim o variabila in care vom memora aceste
     coordonate. }
   drum : TListaCoordonate;
   { Mai definim o variabila care sa memoreze numarul de pasi efectuati
     de cal pe tabla. }
   nr_pasi : Integer;

   { Calul se deplaseaza pe tabla de sah prin sarituri in forma de L.
     Ca urmare atunci cand el paraseste o casuta oarecare de pe tabla,
```

```
      el poate face opt mutari pentru a ajunge intr-o alta casuta.
      Se pune problema cum calculam care sunt cele opt casute in care
      ar putea ajunge calul.
      O solutie la indemana este sa pastram "delta"-liniile si
      "delta"-coloanele pentru fiecare din cele opt mutari posibile. Adica
      sa spunem, pentru fiecare din cele opt mutari, cu cate linii si cu
      cate coloane trebuie sa se deplaseze calul pentru a ajunge in
      noua casuta. }
   mutari : TListaCoordonate;
   { Mai pastram o variabila care sa ne zica numarul de mutari posibile.
     E clar ca acest numar de mutari va fi 8, dar il pastram ca si variabila
     si nu scriem "8" direct in program. }
   nr_mutari : Integer;

   { Pastram o variabila booleana pe care o vom seta pe "true" in momentul
     in care am gasit un drum pentru cal. In felul acesta vom sti sa oprim
     cautarea dupa ce se gaseste primul drum. }
   drum_gasit : Boolean;


{ In aceasta procedura definim care sunt casutele perforate si
  apoi initializam matricea care va pastra configuratia tablei de sah. }
procedure Initializare;
var i, j : Integer;
begin
     { Inainte de toate initializam vectorul de "delta"-linii si
       "delta"-coloane cu ajutorul caruia vom calcula cele opt
       mutari posibile pentru cal. }
     nr_mutari := 8;
     mutari[1][1] := -1; mutari[1][2] := -2;
     mutari[2][1] := -2; mutari[2][2] := -1;
     mutari[3][1] := -2; mutari[3][2] :=  1;
     mutari[4][1] := -1; mutari[4][2] :=  2;
     mutari[5][1] :=  1; mutari[5][2] :=  2;
     mutari[6][1] :=  2; mutari[6][2] :=  1;
     mutari[7][1] :=  2; mutari[7][2] := -1;
     mutari[8][1] :=  1; mutari[8][2] := -2;

     { Apoi definim casutele perforate. Vom avea un numar de 8 casute
       perforate. In mod normal aceste casute perforate s-ar citi din
       fisier, dar pentru a simplifica programul noi le definim direct
       din cod. }
     nr_perf := 8;
     perforatii[1][1] := 1; perforatii[1][2] := 5;
     perforatii[2][1] := 3; perforatii[2][2] := 2;
     perforatii[3][1] := 3; perforatii[3][2] := 5;
     perforatii[4][1] := 4; perforatii[4][2] := 4;
     perforatii[5][1] := 5; perforatii[5][2] := 4;
     perforatii[6][1] := 6; perforatii[6][2] := 1;
     perforatii[7][1] := 6; perforatii[7][2] := 3;
     perforatii[8][1] := 7; perforatii[8][2] := 6;

     { Iar apoi initializam matricea care pastreaza configuratia
       tablei de sah. Initial umplem toata matricea cu 0, care inseamna
       ca nici o casuta nu este perforata. }
     for i := 1 to N do
         for j := 1 to N do
             tabla[i][j] := 0;

     { Si dupa ce toata matricea este plina cu 0, punem 1 in locurile
       unde avem casute perforate. }
```

```
     for i := 1 to nr_perf do
         tabla[ perforatii[i][1] ][ perforatii[i][2] ] := 1;
end;


{ Aceasta procedura afiseaza pe ecran configuratia tablei de sah,
  impreuna cu drumul parcurs de cal pana in momentul afisarii.
  Casutele neperforate sunt marcate cu un '#', iar casutele perforate
  sunt marcate cu un 'o'.
  Drumul calului este reprezentat prin numere. Prima casuta vizitata
  este marcata cu 1, a doua casuta vizitata e marcata cu 2, etc. }
procedure AfiseazaTabla;
var i, j, k : Integer;
    pas : Integer;
begin
     ClrScr;
     { i parcurge toate liniile tablei de sah. }
     for i := 1 to N do
         begin
              { j parcurge toate coloanele tablei de sah. }
              for j := 1 to N do
                  begin
                       { Pentru fiecare casuta de coordonate (i, j)
                         verificam daca ea a fost vizitata de cal. Daca
                         da, atunci pastram si pasul in care a fost
                         vizitata. }
                       pas := -1;
                       for k := 1 to nr_pasi do
                           begin
                                if (drum[k][1] = i) and
                                   (drum[k][2] = j) then
                                   begin
                                        pas := k;
                                        break;
                                   end;
                           end;

                       { Daca am determinat ca aceasta casuta de coordonate
                         (i, j) a fost vizitata de cal, atunci afisam
                         numarul pasului in care a fost vizitata. }
                       if pas <> -1 then
                          Write(pas:3)

                       { Daca nu a fost vizitata, atunci daca nu este
                         perforata afisam un '#'. }
                       else if tabla[i][j] = 0 then
                          Write('#':3)

                       { Iar nu a fost vizitata si e perforata, atunci
                         afisam un 'o'. }
                       else
                          Write('o':3);
                  end;

              { Dupa ce am epuizat casutele de pe o linie a tablei,
                trecem si pe ecran la o linie noua. }
              WriteLn;
         end;

     { Dupa ce am afisat toata tabla de sah, asteptam un anumit interval
       de timp pentru a da ragaz utilizatorului sa interpreteze informatia
```

```
       afisata pe ecran. }
     Delay(10000);
end;


{ Aceasta este procedura propriu-zisa de backtracking. Pentru a o intelege,
  trebuie sa privim lucrurile intr-un mod recursiv:

  Consideram ca am ajuns deja in casuta de coordonate (linie, coloana). Sa
  zicem ca nu ne intereseaza *cum* anume am ajuns pana in aceasta casuta.
  Ce este important e ceea ce vom face mai departe.

  Pai daca am ajuns deja aici, atunci incercam sa mergem mai departe prin
  una din cele opt mutari posibile in forma de L. Pentru aceasta calculam
  pe rand fiecare casuta in care s-ar putea sari si verificam cateva aspecte
  pentru ea:
   - sa se afle pe tabla de sah (dintr-o casuta situata pe marginea tablei
     de sah s-ar putea sari si in afara tablei);
   - sa nu fie perforata;
   - sa nu se mai fi trecut pe acolo inainte.

  Pentru fiecare casuta care indeplineste aceste trei conditii, facem
  saritura in ea, incercam apoi in mode recursiv sa mergem mai departe
  si din noua casuta, dupa care anulam saritura. In felul acesta tatonam
  fiecare din cele opt mutari posibile. Acesta este de fapt mecanismul
  de backtracking. La fiecare pas se merge prin incercare si se tatoneaza
  fiecare posibila mutare.

  Daca la un moment dat se constata ca s-a ajuns la destinatie, atunci
  se opreste cautarea. Se va seta pe "true" o variabila globala care va
  semnala ca e momentul sa se revina din apelurile recursive. }

{ Semnificatia parametrilor "linie" si "coloana" este urmatoarea: pana
  in prezent calul a parcurs un drum oarecare, si acum se pune problema
  de a extinde drumul cu o noua casuta, si anume cu casuta de coordonate
  (linie, coloana). Daca este posibil, casuta (linie, coloana) se va
  adauga la drumul parcurs. }
procedure MutaPiesa(linie, coloana : Integer);
var k, i : Integer;
    linie_urm, coloana_urm : Integer;
    am_mai_trecut_pe_aici : Boolean;
begin
     { Verificam daca nu cumva s-a ajuns in casuta de coordonate (8, 8),
       caz in care cautarea se incheie. }
     if (linie = N) and (coloana = N) then
        begin
             { Daca s-a ajuns la casuta (8, 8), atunci o adaugam la drumul
               parcurs, facem afisarea tablei si iesim din procedura
               recursiva. }
             nr_pasi := nr_pasi + 1;
             drum[nr_pasi][1] := linie;
             drum[nr_pasi][2] := coloana;

             drum_gasit := true;

             AfiseazaTabla;
             WriteLn;
             WriteLn('Am gasit solutie!');

             exit;
        end;
```

```
{ Acum urmeaza partea caracteristica backtracking-ului. Se tatoneaza
  *fiecare* mutare posibila a calului. }
for k := 1 to nr_mutari do
    begin
         { Pentru mutarea curenta, calculam coordonatele casutei
           in care ar trebui sa se sara. }
         linie_urm := linie + mutari[k][1];
         coloana_urm := coloana + mutari[k][2];

         { Verificam daca aceasta casuta este situata pe tabla si
           daca nu este perforata. Acestea doua sunt primele conditii
           pentru a putea face saritura in noua casuta. }
         if (linie_urm >= 1) and (linie_urm <= N) and
            (coloana_urm >= 1) and (coloana_urm <= N) and
            (tabla[linie_urm][coloana_urm] = 0) then
            begin
                 { Daca casuta este situata pe tabla si nu este
                   perforata, mai verificam si daca nu cumva s-a mai
                   trecut prin ea. }
                 am_mai_trecut_pe_aici := false;
                 for i := 1 to nr_pasi do
                     begin
                          if (drum[i][1] = linie_urm) and
                             (drum[i][2] = coloana_urm) then
                             begin
                                  am_mai_trecut_pe_aici := true;
                                  break;
                             end;
                     end;

                 { Daca nici nu s-a mai trecut prin noua casuta,
                   atunci incercam sa facem saritura in ea in
                   speranta ca vom gasi astfel un drum catre casuta
                   (8, 8). }
                 if not(am_mai_trecut_pe_aici) then
                    begin
                         { Adaugam casuta la drumul parcurs. }
                         nr_pasi := nr_pasi + 1;
                         drum[nr_pasi][1] := linie;
                         drum[nr_pasi][2] := coloana;

                         { Facem o afisare a drumului de pana acum. }
                         AfiseazaTabla;

                         { In mod recursiv incercam sa mutam mai
                           departe piesa. }
                         MutaPiesa(linie_urm, coloana_urm);

                         { Daca cumva s-a gasit un drum, atunci oprim
                           cautarea. Nu mai este nevoie sa mai tatonam
                           alte mutari posibile. }
                         if drum_gasit then
                            Break

                         { Daca cumva nu s-a gasit inca un drum, atunci
                           mutarea pe care am facut-o se pare ca nu
                           este utila. Deci o anulam si vom incerca
                           cu celelalte mutari posibile netatonate. }
                         else
```

```
                                        begin
                                             { Stergem casuta din drumul parcurs. }
                                             drum[nr_pasi][1] := 0;
                                             drum[nr_pasi][2] := 0;
                                             nr_pasi := nr_pasi - 1;

                                             { Si facem o afisare a tablei. }
                                             AfiseazaTabla;
                                        end;
                              end;
                    end;
          end;
end;

{ Programul principal. }
begin
     ClrScr;

     { Facem initializarile din program. }
     Initializare;

     { Setam pe "false" variabila care ne indica daca s-a gasit solutie. }
     drum_gasit := false;

     { Si incepem cautarea tatonand casuta (1, 1). }
     MutaPiesa(1, 1);

     { Daca dupa ce cautarea se incheie nu avem nici un drum gasit,
       afisam un mesaj corespunzator. }
     if not(drum_gasit) then
        begin
             { Afisam configuratia tablei pentru a o putea analiza. }
             AfiseazaTabla;
             WriteLn;
             WriteLn('Nu exista solutie!');
        end;
end.
```