

## Problème I - EXCLUSION

20 points

```

program EXCLUSION;

var P,Q,M,N,I : integer;

function CONT (X,Y : integer) : boolean;
var TEMP : boolean;
begin
  TEMP := false;
  while X > 0 do
    begin
      if X mod 10 = Y
      then TEMP := true;
      X := X div 10
    end;
  CONT := TEMP
end;

function PREMIER (X : integer) : boolean;
var TEMP : boolean;
  I : integer;
begin
  TEMP := true;
  for I := 2 to round(sqrt(X)) do
    if X mod I = 0
    then TEMP := false;
  if X = 1
  then TEMP := false;
  PREMIER := TEMP
end;

function SOMME (X : integer) : integer;
var SUM : integer;
begin
  SUM := 0;
  while X > 0 do
    begin
      SUM := SUM + X mod 10;
      X := X div 10
    end;
  SOMME := SUM
end;

function SOMMEPREMIER (X : integer) : boolean;
begin
  SOMMEPREMIER := PREMIER(SOMME(X))
end;

function EXCLU (X,M,N : integer) : boolean;
begin
  EXCLU := (X mod M = 0) or
           (X mod N = 0) or
           CONT(X,M) or
           CONT(X,N) or
           PREMIER(X) or
           SOMMEPREMIER(X)
end;

begin
  readln(P,Q,M,N);
  for I := P to Q do
    if not EXCLU(I,M,N)

```

```

        then write(I, ' ');
    writeln;
    readln
end.

```

## Problème II - CODAGE

30 points

```

program SPIONAGE;

var CLEF, TEXTE, TEXTOUT : string;
    TABLE : array ['a'..'z'] of char;
    FIN, FOUT : text;
    ACTION : char;
    STR : string;

function VALID (CLEF : string) : boolean;
var TEMP : boolean;
    STR : string;
begin
    TEMP := true;
    while length(CLEF) > 0 do
        begin
            STR := copy(CLEF, 1, 1);
            delete(CLEF, 1, 1);
            if pos(STR, CLEF) > 0
            then TEMP := false
            end;
        VALID := TEMP
    end;

procedure DOTABLE (CLEF : string);
var P, C : char;
    I : integer;
begin
    P := 'a';
    for I := 1 to length(CLEF) do
        begin
            TABLE[P] := CLEF[I];
            P := succ(P)
        end;
    for C := 'z' downto 'a' do
        if pos(C, CLEF) = 0
        then begin
            TABLE[P] := C;
            P := succ(P)
        end
    end;

function LOOKTABLE (CH : char) : char;
var I, RES : char;
begin
    for I := 'a' to 'z' do
        if TABLE[I] = CH
        then RES := I;
    LOOKTABLE := RES
end;

procedure CODE (ACTION : char; TEXT : string;
                var TEXTOUT : string);
var I : integer;
begin
    TEXTOUT := '';
    for I := 1 to length(TEXTE) do
        if ACTION = 'c'
        then if TEXTE[I] in ['a'..'z']
            then TEXTOUT := TEXTOUT + TABLE[TEXTE[I]]
            else TEXTOUT := TEXTOUT + TEXTE[I]
        else if TEXT[I] in ['a'..'z']

```

```

        then TEXTOUT := TEXTOUT + LOOKTABLE (TEXTE[I])
        else TEXTOUT := TEXTOUT + TEXTE[I]
end;

begin
  assign (FIN, 'CODEIN.TXT');
  reset (FIN);
  readln (FIN, CLEF);
  readln (FIN, TEXTE);
  close (FIN);
  assign (FOUT, 'CODEOUT.TXT');
  rewrite (FOUT);
  if VALID (CLEF)
  then begin
    writeln ('Codage / decodage');
    writeln ('=====');
    write ('Coder/decoder (C/D) : ');
    readln (ACTION);
    DOTABLE (CLEF);
    if ACTION in ['c', 'C']
    then CODE ('c', TEXTE, TEXTOUT)
    else CODE ('d', TEXTE, TEXTOUT);
    writeln (FOUT, TEXTOUT)
  end
  else writeln (FOUT, 'mot secret non-valide');
  close (FOUT)
end.

```

## Problème III - SOLITAIRE

50 points

Le troisième problème, le jeu du solitaire sur une planche triangulaire, peut être résolu par la méthode du backtracking. Cette méthode consiste à essayer toutes les possibilités et à revenir en arrière si aucun coup n'est plus possible. A partir de la on essaye alors une autre possibilité.

La méthode du backtracking est traitée à l'aide d'exemples, dans le livre du célèbre professeur Niklaus Wirth, créateur des langages de programmation Pascal et Modula.

L'étude de ce livre est vivement recommandée aux élèves candidats du concours et aux participants à l'olympiade internationale en informatique, l'IOI.

On y étudiera, entre autres, le problème des huit dames sur l'échiquier et celui de la course du cavalier, qui doit visiter toutes les cases.

Surtout ce dernier problème est très proche de notre exercice du solitaire.

Voici la solution modèle en langage Pascal:

```

program Solitaire;

type coord = record
    fromx, fromy, tox, toy : integer
end;

var a, b : array [1..6] of integer;      { possible moves }
    h : array [1..5, 1..5] of integer;  { the board }
    hx, hy : integer;                  { initial hole }
    result : array [1..50] of coord;    { result array }
    r : integer;                       { result array counter }
    fin, fout : text;                  { in and out files }

procedure setup;
var i, j : integer;
begin
    { possible moves array }
    a[1] := -2; b[1] := 0;
    a[2] := 0; b[2] := 2;

```

```

a[3] := 2; b[3] := 2;
a[4] := 2; b[4] := 0;
a[5] := 0; b[5] := -2;
a[6] := -2; b[6] := -2;
{ put the pegs on the board }
for i := 1 to 5 do
  for j := 1 to i do
    h[i,j] := 1;
{ get the initial hole coords }
assign(fin,'SOLITIN.TXT');
reset(fin);
read(fin,hx,hy);
close(fin);
{ remove peg of initial hole }
h[hx,hy] := 0;
{ initialize result array counter }
r := 0;
{ prepare out file }
assign(fout,'SOLITOUT.TXT');
rewrite(fout)
end;

function ok (u,v,i,j : integer) : boolean;
var tempok : boolean;
begin
  tempok := true;
  { are we still on board ? }
  case u of
    1 : tempok := tempok and (v in [1]);
    2 : tempok := tempok and (v in [1,2]);
    3 : tempok := tempok and (v in [1,2,3]);
    4 : tempok := tempok and (v in [1,2,3,4]);
    5 : tempok := tempok and (v in [1,2,3,4,5])
  else
    tempok := false
  end;
  { is the target hole empty ? }
  { and is there a peg to jump over ? }
  tempok := tempok and (h[u,v] = 0)
                and (h[(u+i) div 2, (v+j) div 2] = 1);
  ok := tempok
end;

function fini : boolean;
var i,j,pegs : integer;
    lastx,lasty : integer;
begin
  pegs := 0;
  for i := 1 to 5 do
    for j := 1 to i do
      if h[i,j] = 1
      then begin
        pegs := pegs + 1;
        lastx := i;
        lasty := j
      end;
  { is there only one peg remaining ? }
  { and is the last peg in the initial hole ? }
  fini := (pegs = 1) and (lastx = hx) and (lasty = hy)
end;

procedure trymove;
var k : integer;      { possible moves array counter }
    i,j : integer;    { starting coords }
    u,v : integer;    { target coords }
begin
  for i := 1 to 5 do
    for j := 1 to i do
      if h[i,j] <> 0   { try the pegs, not the holes }

```

```

then begin
    k := 0;
    repeat
        k := k + 1;
        u := i + a[k]; v := j + b[k];
        if ok(u,v,i,j)
            then begin
                { we found a valid move, so remove }
                { the jumping and the jumped-over peg }
                h[i,j] := 0;
                h[(u+i) div 2, (v+j) div 2] := 0;
                { put a peg in the target hole }
                h[u,v] := 1;
                { update result array }
                r := r + 1;
                with result[r] do
                    begin
                        fromx := i; fromy := j;
                        tox := u; toy := v
                    end;
                { start recursion ! }
                trymove;
                if not fini
                    then begin
                        { backtracking !
                        the last move wasn't ok, so undo it
                        replace the jumping and the jumped-over
                        peg on the board }
                        h[i,j] := 1;
                        h[(u+i) div 2, (v+j) div 2] := 1;
                        { remove the peg from the target hole }
                        h[u,v] := 0;
                        { update result array }
                        r := r - 1
                    end
                end
            end
        { all 6 directions tried ? }
    until k = 6
    { well, then try next peg }
end
end;

procedure writeout;
var i : integer;
begin
    for i := 1 to r do
        with result[i] do
            begin
                write(fout, fromx, ' ', fromy);
                write(fout, ' -> ');
                writeln(fout, tox, ' ', toy);
            end
        end
    end;
end;

begin
    setup;
    trymove;
    if r > 0
        then writeout
        else writeln(fout, 'No solution!');
    close(fout)
end.

```

Remarquons qu'avec le trou initial dans une position intérieure (3,2 ou 4,2 ou 4,3) il n'y a pas de solution.