The IOI'97 Competition

NOTE: The provided solutions do not meet my quality standards.

Some should not even have scored many points. -Tom Verhoeff

Programme: Mars.exe

```
Authors: Kevin Dennis and Ashton Mason
Language: C/C++
Algorithm:

The programme reads the terrain into an array.
A binary tree is then created for all possible
  movements from the start to the end.
Thus.... Only complete paths can be seen as the
  tree does not continue past rough terrain.
Therefore.... The paths are evaluated to see which
  sample the most times, and the larges sum of the
  paths according to how many MEV's there are, is
  chosen and these paths are thenused as output.
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "terrain.h"


#define EAST 1
#define NORTH 0

#define CLEAR 0
#define ROUGH 1
#define SAMPLE 2


int numberOfVehicles;
int xMax, yMax;
terrain land;


        // read the input data file and create the terrain

void readDataFile(char *fileName)
{
  FILE *file = fopen(fileName, "r");
  if (!file)
  {
    printf("can't open input file\n");
    exit(1);
  }

  fscanf(file, " %d", &numberOfVehicles);
  fscanf(file, " %d %d", &xMax, &yMax);
  land.create(xMax, yMax);

  for (int y = 1; y <= yMax; y++)
  {
    for (int x = 1; x <= xMax; x++)
    {
      int groundType;
      fscanf(file, " %d", &groundType);
      land.setGroundType(x, y, groundType);
    }
  }

  fclose(file);
}


        // finds the best path from (x, y) to the base at (xMax, yMax)
        // (in terms of total number of rock samples collected)
        // counts the sample on (x, y)
        // returns -1 if no path exists

int findBestPath(int x, int y)
{
  int thisSample;

        // calculate the value of this position

  switch (land.getGroundType(x, y))
  {
    case CLEAR:
      thisSample = 0;          // no sample here
    break;
    case ROUGH:
      return -1;               // no possible route
    break;
    case SAMPLE:
      thisSample = 1;          // sample here
    break;
  }

        // check for terminating condition: already at base

  if (x == xMax && y == yMax)
  {
    return thisSample;
  }

        // find the best route from here
        // (can either go north or east)

  int northValue = -1;
  int eastValue = -1;

        // consider moving north, if we can

  if (y < yMax)
  {
    northValue = findBestPath(x, y + 1);
  }

        // consider moving east, if we can

  if (x < xMax)
  {
    eastValue = findBestPath(x + 1, y);
  }
```

```
        // if neither is possible, return impossible for (x, y)

  if (northValue == -1 && eastValue == -1)
  {
    return -1;
  }

        // choose whichever is better, north or east
        // mark which one we chose for this square
        // return the total number of samples for that route
        // plus the sample value for this spot

  if (eastValue > northValue)
  {
    land.setChosenDirection(x, y, EAST);
    return thisSample + eastValue;
  }
  else
  {
    land.setChosenDirection(x, y, NORTH);
    return thisSample + northValue;
  }
}


        // control the vehicles and write the output file

void writeOutputFile(char *fileName)
{
  int rover;

  FILE *file = fopen(fileName, "w");
  if (!file)
  {
    printf("can't open output file\n");
    exit(1);
  }

        // for each rover

  for (rover = 1; rover <= numberOfVehicles; rover++)
  {

        // find the optimal path for this rover
        // from the start to the base, given the current
        // grid situation
        // this process marks the best path in the land array
        // so that we can follow it later
        // if we can't find any path at all then we give up

    if (findBestPath(1, 1) == -1)
    {
      break;
    }

        // follow the trail marked for this rover
        // move from (1, 1) to (xMax, yMax)
        // output to the file and mark the trail as used up
        // as we go

    int x = 1;
    int y = 1;
    while (x != xMax || y != yMax)
    {

        // mark this square as used up (now clear)

      land.setGroundType(x, y, CLEAR);

        // find out which direction lies along the optimal path
        // found above

               if (land.getChosenDirection(x, y) == NORTH)
               {
                 fprintf(file, "%d 0 \n", rover);
                 y++;
               }
               else
               {
                 fprintf(file, "%d 1 \n", rover);
                 x++;
      }
         }

        // mark the final square as used up

    land.setGroundType(x, y, CLEAR);
  }

  fclose(file);
}


        // the main routine
        // reads the input file and writes the output file
        // if an input filename is passed, it is used,
        // otherwise MARS.DAT is assumed

int main(int argc, char **argv)
{
  char fileName[100];

  strcpy(fileName, "MARS.DAT");
  if (argc >= 2)
  {
         strcpy(fileName, argv[1]);
  }

  readDataFile(fileName);
  writeOutputFile("MARS.OUT");
  printf("\n\n - - The White Knight - - \n     - Kevin Dennis -\n");
  return 0;
}
```

//Game of Hex: CPP, Michael Nelte

```
#include <iostream.h>
#include <conio.h>
#include "hexcpp.h"

int max;


void display(void)
{
        int row, col, l;
        gotoxy (1, 1);
        for (row=1; row<=max; row++)
        {
                for (col=1; col<row; col++)
                        cprintf (" ");
                for (col=1; col<=max; col++)
                 { l = LookAtBoard (max+1-row, col);
                        textcolor (l == 0 ? 15 : l*3-2);
                        cprintf ("%d ",l);
                 }
                textcolor (WHITE);
                cprintf ("\r\n");
        }
}

int x, y;

int mayuse(int x, int y)
{
  if (x < 1) return 5;
  if (y < 1) return 5;
  if (y > max) return 5;
  if (x > max) return 5;
  return LookAtBoard (x+1, y+1);
}

int connected(int x,int y)
{
  if (LookAtBoard(x, y) != 0) return 0;
  if (LookAtBoard(x+1, y+1) == 1) return 1;
  if (LookAtBoard(x+1, y) == 1) return 1;
  if (LookAtBoard(x, y+1) == 1) return 1;
  if (LookAtBoard(x-1, y) == 1) return 1;
  if (LookAtBoard(x, y-1) == 1) return 1;
  if (LookAtBoard(x-1, y-1) == 1) return 1;
  return 0;
}


PickBlock()
{
  int row, col, l;
  for(row=1;row <=max;row++)
  {
     int spot;
     if (spot%2==0)
       spot = max/2 + row/2;
     else
       spot = max/2 - row/2;
     spot = max/2+row;
     if (spot > max)
     {
       spot = max -row+1;
     }
     if (LookAtBoard(spot, spot) == 0)
     {
       x = spot;
       y = spot;
       return 0;
     }
  }

  l = connected(x+1, y+1);
  if (l)
  {
    x++;
    y++;
    return 0;
  }
  for (row=1; row<=max; row++)
  {
    for (col=1; col<=max; col++)
    { l = LookAtBoard (row, col);
      if (l == 0)
      {
        x = row;
        y = col;
        if (connected(x+1, y+1))
        {
          x++;
          y++;
          return 0;
        }
      }
    }
  }

  for (row=1; row<=max; row++)
  {
    for (col=1; col<=max; col++)
    { l = LookAtBoard (row, col);
      if (l == 0)
      {
        x = row;
        y = col;
        return 0;
      }
    }
  }
  return 0;
}

int main (void)
{
  clrscr ();
  max = GetMax ();
  x=0;y=0;
  int r, c, g;
  display ();
//  getval();
  while(!GameIsOver())
  {
        display();
        PickBlock();
        PutHex(x, y);
        MakeLibMove();
  }
  display();

  return 0;
}
```

Program: Toxic

Author: Michael Nelte and Kevin Dennis
Language: C++
Algorithm:

A 3d Array is created to represent the "fruit" which the iShongololo is going to be eating. This array is then initialised to all being edible.

The iShongololo starts eating at the block 1,1,1 and then moves to the first block eaten. It then moves in a positive lengthwise direction eating all adjacent blocks (horizontal and vertical) that don't violate the rules.

On reaching a side, it calculates whether the iShongololo can turn without violateing rules and stops eating adjacent blocks. It will then turn and continue this process. When no further horizonatal blocks can be eaten, the iShongololo will move downwards 4 blocks and and continue this pattern, noting that it also eats as many blocks upwards and downwards and horizantally as possible, again, leaving enough uneaten block at the corners to turn.

Once there are no further edible blocks, the programme ends.

To further optimise the solutions, the solutions are run from 3 directions, alternately taking the length, width and height as the 'top' face for the iShongololo to use. The best of these is then used.

```
#include <fstream.h>
#include <iostream.h>
#include <string.h>
#include <stdlib.h>

#define EMPTY 0
#define FOOD  1
#define BODY  2
#define TOXIC 3

#define EAST  0
#define WEST  1
#define SOUTH 2
#define NORTH 3
#define DOWN  4
#define UP    5


class Toxic {
  private:
    char*** block;
    int sizex, sizey, sizez;
    int Tsizex, Tsizey, Tsizez;
    int Tstartx, Tstarty, Tstartz;

    ofstream out;
    int posx, posy, posz;
    int order[6];
    int d;
    int majordir;
    int count;
    int axisorder;
  public:
    Toxic(char* t, int);
    ~Toxic();
    int run();
    void Eat();
    int Move(int dir);
    void OppositeDir(int dir);
    void Mark();
    int Free() {return block[posx][posy][posz] == FOOD;}
    void Out();
    int ShouldEat();
    int InGood();


};

typedef char** charss;
typedef char* chars;

Toxic::Toxic(char* t, int i)
:axisorder(i)
{
  char name[50];
  char fred[100];
  itoa(i, fred, 10);
  strcpy(name, "toxic.");
  strcat(name, fred);
  ifstream in(t);
  out.open(name);
  switch (i)
  {
    case 0: in >> sizex >> sizey >> sizez; break;
    case 1: in >> sizex >> sizez >> sizey; break;
    case 2: in >> sizey >> sizex >> sizez; break;
    case 3: in >> sizey >> sizez >> sizex; break;
    case 4: in >> sizez >> sizey >> sizex; break;
    case 5: in >> sizez >> sizex >> sizey; break;
    default: return;
  }
  Tstartx = 1;
  Tstarty = 1;
  Tstartz = 1;
  Tsizex  = sizex;
  Tsizey  = ((sizey-1)/3)*3+1;
  Tsizez  = sizez;
  count = 1;


  block = new charss[sizex+2];
  for(i=0;i<sizex+2;i++)
  {
    block[i] = new chars[sizey+2];
    for(int j = 0; j < sizey+2; j++)
    {
      block[i][j] = new char [sizez+2];
      for (int k = 0; k < sizez+2; k++)
       block[i][j][k] = EMPTY;
    }
  }
  for (i = 1; i <= sizex; i++)
    for(int j = 1; j <= sizey; j++)
      for(int k = 1; k <= sizez; k++)
       block[i][j][k] = FOOD;
  for(i=0;i<6;i++)
   order[i] = i;
}

Toxic::~Toxic()
{
  for(int i=0;i<sizex+2;i++)
  {
    for(int j = 0; j < sizey+2; j++)
      delete[] block[i][j];
    delete[] block[i];
  }
  delete[] block;

}

void Toxic::Out()
{
  switch (axisorder)
  {
    case 0: out << posx << " " << posy << " " << posz; break;
    case 1: out << posx << " " << posz << " " << posy; break;
    case 2: out << posy << " " << posx << " " << posz; break;
    case 3: out << posy << " " << posz << " " << posx; break;
    case 4: out << posz << " " << posy << " " << posx; break;
    case 5: out << posz << " " << posx << " " << posy; break;
    default: return;
  }


}

void Toxic::Eat()
{
  Out();
  Mark();
  for (int d=EAST; d <= UP; d++)
  {
    if (Move(d))
    {
      if (Free())
       Mark();
    }
    OppositeDir(d);
  }
}

void Toxic::Mark()
{
  block[posx][posy][posz] = TOXIC;
}

int Toxic::Move(int dir)
{
  dir = order[dir];
  switch(dir) {
    case EAST : posx++; break;
    case SOUTH: posy++; break;
    case WEST : posx--; break;
    case NORTH: posy--; break;
    case DOWN : posz++; break;
    case UP   : posz--; break;
  };
  if (posx > sizex || posx < 1 ||
      posy > sizey || posy < 1 ||
      posz > sizez || posz < 1)
      return 0;     // Bad Move
  return 1;
}

void Toxic::OppositeDir(int dir)
{
  dir = order[dir];
  switch(dir) {
    case EAST : posx--; break;
    case SOUTH: posy--; break;
    case WEST : posx++; break;
    case NORTH: posy++; break;
    case DOWN : posz--; break;
    case UP   : posz++; break;
  };
}

int Toxic::ShouldEat()
{
  if (order[0] == EAST && posx == Tsizex-1 && order[d] == majordir) return 0;
  if (order[0] == WEST && posx == Tstartx+1 && order[d] == majordir) return 0;
  if (order[d] == DOWN && posx == Tstartx+1 && posy == Tsizey && majordir == SOUTH)
return 0;
  if (order[d] == DOWN && posx == Tsizex-1 && posy == Tsizey && majordir == SOUTH) return
0;
  if (order[d] == DOWN && posx == Tstartx+1 && posy == 1 && majordir == NORTH) return 0;
  if (order[d] == DOWN && posx == Tsizex-1 && posy == 1 && majordir == NORTH) return 0;
  return 1;
}

int Toxic::InGood()
{
  if (posx > Tsizex || posx < Tstartx ||
      posy > Tsizey || posy < Tstarty ||
      posz > Tsizez || posz < Tstartz)
      return 0;     // Bad Move
  return 1;
}

int Toxic::run()
{
  majordir = SOUTH;
  int dir;
  posx = 1;
  posy = 1;
  posz = 1;
  int lastdir = 0;
  out << "E 1 1 1\nM 1 1 1";
  Mark();
  int eat=0;
  int southcount = 0;
```

```
do
{
  dir = -1;
  for (d=EAST; d <= UP; d++)
  {
    if (Move(d))
    {
      if (Free())
      {
        if (dir == -1 && InGood()) dir = d;
        else
        {
          if (ShouldEat())
          {
            if (!eat || !(order[d] == EAST || order[d] == WEST))
            {
              out << "\nE ";
              Eat();
              count++;
            }
          }
          else
            eat = 1;
        }
      }
    }
    OppositeDir(d);
  }
  if (dir == -1) break;
  else
  {
    for (d=EAST; d <= UP; d++)
    {
      if (Move(d))
      {
        if (Free())
          Mark();
      }
      OppositeDir(d);
    }
    Move(dir);
    Mark();
    count++;
    out << "\nE ";
    Out();
    out << "\nM ";
    Out();

    if (((posy == sizey && order[dir] == SOUTH) ||
        (posy == 1 && order[dir] == NORTH))
        &&
        sizey%3==0 && (posx == 1 || posx == sizex))
    {
      eat = 1;
      if (order[0] != DOWN)
      {
        int t;
        t = order[0];
        order[0] = order[1];
        order[1] = t;
        t = order[0];
        order[0] = order[4];
        order[4] = t;
        }
        southcount = 0;
      }
      else
      if (order[dir] == majordir)
      {
        eat = 1;
        if (order[0] != majordir)
        {
          int t = order[0];
          order[0] = order[1];
          order[1] = order[2];
          order[2] = t;
        }
        southcount++;
        if (southcount >= 3)
        {
          int t = order[0];
          order[0] = order[2];
          order[2] = t;
          eat = 0;
          southcount = 0;
        }
      }

      if (order[dir] == DOWN)
      {
        eat = 1;
        if (order[0] != DOWN)
        {
          int t;
          t = order[0];
          order[0] = order[1];
          order[1] = t;
          t = order[0];
          order[0] = order[4];
          order[4] = t;
        }
        southcount++;
        if (southcount > 3)
        {
          int t = order[0];
          order[0] = order[4];
          order[4] = t;
          eat = 0;
          southcount = 0;
        }
      }


      if (order[dir] == DOWN && lastdir != dir)
      {
        int t = order[2];
        order[2] = order[3];
        order[3] = t;
        majordir = order[2];
      }
      lastdir = dir;
    }
  }while(1);
//  out << "\n";
  cout << "\n\nBlocks eaten: " << count  << endl;
  out.close();
  return count;
}


int main(void)
{
  int count=0;
  int maxcount=0;
  int run=0;
  int i=0;
  for(;i<6;i++)
  {
    Toxic t("toxic.dat", i);
    count = t.run();
    if (count > maxcount)
    {
      maxcount = count;
      run = i;
    }
  }
  char name[100];

  char fred[100];
  itoa(run, fred, 10);
  strcpy(name, "copy toxic.");
  strcat(name, fred);
  strcat(name, " toxic.out");
  system(name);
//  cout << name << endl;

  cout << "Best count: " << maxcount << endl;

  cout << "\n\n - - The White Knight - - & - -     Fred!     - -\n     - Kevin Dennis -
- Michael Nelte -\n";
  return 0;
}
```

Map labelling

Author: Michael Nelte
Date: 24 November 1997
Language: C++
Algorithm:

* The lables are sorted in order of smallest first.
* Each lable in the above order is inserted one after the other if it can be in one of the 4 positions possible.
* Once all the lables that can be inserted this way have been then the lables that are already in get shifted arround if they can be. This might open gaps into which uninserted lables are placed.

There are two versions of maps (maps1 and maps2). This is because many contestants used the sample solution in their programming directory to determine direction, and this sample was wrong. One version corresponds to the problem description and the other to the sample solution.

```
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <stdlib.h>

//#define DEBUG

class city {
  public:
    int x, y, divx, divy;
    int posx, posy;
    int pos;
    long order;
    int position;
    city(){}
    void set(char* n, int p, int X, int Y, int sizeX, int sizeY);
};

void city::set(char* n, int p, int X, int Y, int sizeX, int sizeY)
{
  order = (long)X + Y + (sizeX* sizeY)*1000l;
//  order = random(1000);
  pos = p;
  posx = -1;
  posy = -1;
  x = X;
  y = Y;
  divx = (strlen(n)+1)*sizeX;
  divy = sizeY;
//  order += (long)divx*divy*1000000;
//  order = random(100000);
}


class map {
  private:
    char* filename;
    ifstream in;
    int num;
    city* fred;
    public : int count;

  public:
    map(char* name);
    ~map();
    void run();
    void sortorder();
    void sort();
    int used(int which, int x1, int y1, int x2, int y2);
    void insert(int i);
    void move(int i);
};

map::map(char* t)
//:filename(t), count(-0)
{
  filename = t;
  char name[100];
  strcpy(name, filename);
  strcat(name, ".dat");
  in.open(name);
  in >> num;
  fred = new city[num];
  int i;
  for(i =0 ; i< num; i++)
  {
    int x, y, sizex, sizey;
    char name[300];
    in >> x >> y>> sizex>> sizey>>name;
    fred[i].set(name, i, x, y, sizex, sizey);
  }
}

map::~map()
{
  char name[100];
  strcpy(name, filename);
  strcat(name, ".out");
  ofstream out(name);
  for (int i = 0; i < num; i++)
     out << fred[i].posx << " " << fred[i].posy << endl;
  delete[] fred;
}

void map::sort()
{

  int pos;
  for (int i = 0; i < num; i++)
  {
    pos = i;
    for (int j = i+1; j < num; j++)
       if (fred[j].order < fred[pos].order) pos = j;
    if (j != i)
    {
      city t;
      t = fred[i];
      fred[i] = fred[pos];
      fred[pos] = t;
    }
  }
```

```
}

int map::used(int which, int x1, int y1, int x2, int y2)
{
  if (x1 < 0) return 1;
  if (x2 > 1000) return 1;
  if (y1 < 0) return 1;
  if (y2 > 1000) return 1;

  int i;
  for (i = 0; i < num; i++)
  {
    if (i == which) continue;            // may use own space
    if (fred[i].x < x2 && fred[i].x >= x1
      && fred[i].y < y2 && fred[i].y >= y1)
        return 3;  // covers city
    if (fred[i].posx != -1)
    if (y1 < fred[i].posy + fred[i].divy
            && y2 > fred[i].posy
            && x1 < fred[i].posx + fred[i].divx
            && x2 > fred[i].posx)
            return 2;        // covers a lable
  }
  return 0;
}

void map::insert(int i)
{
  if (!used(i, fred[i].x-fred[i].divx,     // bottomleft
        fred[i].y+1,
        fred[i].x,
        fred[i].y+fred[i].divy+1))
  {
    fred[i].posx = fred[i].x-fred[i].divx;
    fred[i].posy = fred[i].y+1;
    count++;
    fred[i].position = 2;
  }
  else
  if (!used(i, fred[i].x+1,               // bottomright
        fred[i].y+1,
        fred[i].x+fred[i].divx+1,
        fred[i].y+fred[i].divy+1))
  {
    fred[i].posx = fred[i].x+1;
    fred[i].posy = fred[i].y+1;
    count++;
    fred[i].position = 4;
  }
  else
  if (!used(i, fred[i].x-fred[i].divx,       // topleft
        fred[i].y-fred[i].divy,
        fred[i].x,
        fred[i].y))
  {
    fred[i].posx = fred[i].x-fred[i].divx;
    fred[i].posy = fred[i].y-fred[i].divy;
    count++;
    fred[i].position = 1;
  }
  else
  if (!used(i, fred[i].x+1,                    // topright
        fred[i].y-fred[i].divy,
        fred[i].x + fred[i].divx+1,
        fred[i].y))
  {
    fred[i].posx = fred[i].x+1;
    fred[i].posy = fred[i].y-fred[i].divy;
    count++;
    fred[i].position = 3;
  }
}

void map::move(int i)
{
  int loop = 0;

  switch (fred[i].position)
  {
  case 4:
start:
    if (!used(i, fred[i].x-fred[i].divx,        // topleft
        fred[i].y-fred[i].divy,
        fred[i].x,
        fred[i].y))
    {
      fred[i].posx = fred[i].x-fred[i].divx;
      fred[i].posy = fred[i].y-fred[i].divy;
      fred[i].position = 1;
      break;
    }
  case 1:
    if (!used(i, fred[i].x-fred[i].divx,      // bottomleft
        fred[i].y+1,
        fred[i].x,
        fred[i].y+fred[i].divy+1))
    {
      fred[i].posx = fred[i].x-fred[i].divx;
      fred[i].posy = fred[i].y+1;
      fred[i].position = 2;
      break;
     }
  case 2:
    if (!used(i, fred[i].x+1,                     // topright
        fred[i].y-fred[i].divy,
        fred[i].x + fred[i].divx+1,
        fred[i].y))
    {
      fred[i].posx = fred[i].x+1;
      fred[i].posy = fred[i].y-fred[i].divy;
      fred[i].position = 3;
      break;
    }
  case 3:
    if (!used(i, fred[i].x+1,                // bottomright
        fred[i].y+1,
        fred[i].x+fred[i].divx+1,
        fred[i].y+fred[i].divy+1))
    {
      fred[i].posx = fred[i].x+1;
      fred[i].posy = fred[i].y+1;
      fred[i].position = 4;
      break;
    }
    if (loop == 0)
    {
      loop++;
      goto start;
    }
  }

}

void map::run()
{
  if (count == num) return;
 // cout << "\n Running \n";
  for (int i = 0 ; i < num; i++)
  {
    if (fred[i].posx == -1)
      insert(i);
    else
      move(i);
  }
}

void map::sortorder()  // sort back to correct order
{

  int i;
  city* t = new city[num];

  for(i = 0; i < num; i++)
    t[fred[i].pos] = fred[i];

  delete[] fred;
  fred = t;

}

int main()
{
  randomize();
  cout << "\nStart Michael Nelte's Map Labelling\n\n";
  map themap("maps");

  themap.sort();                          // use secret order for insertion
  themap.count = 0;
  for (int i=0;i<20;i++)    // set the 5 to a larger number for better aprox
    themap.run();
  themap.sortorder();                     // return to orig order
  cout << "\n\n";
  return 0;
}
```

JJ Combrink

Image solution algorithm

```
PROCEDURE Compare
{
 LOOP font from character 1 thru 27
 {
  LOOP through all possible ommitions or duplications of character (41)
  {
   compare the mutated character to the data at position on screen and
   count the number of differences
  }
 CHAR=character with least number of difference
}

{
 Set screen offset to zero
 RUN Compare
 OUTPUT CHAR

 LOOP until end of screen
 {
  Increment screen offset by 19
  LOOP from 1 to 3
  {
   RUN Compare
   Increment screen offset by 1
   OUTPUT the CHAR with least number of differences
  }
 }
}
```

```
{Solution IOI '97 Image recognition
 20-11-97
 JJ Combrink

 Input files from current directory:
 IMAGE.DAT
 FONT.DAT
 Output files to current directory:
 IMAGE.OUT
}


{$r-}
USES Crt,Dos;

CONST ImageFilename='IMAGE.DAT';
      FontFilename='FONT.DAT';
      OutputFilename='IMAGE.OUT';

VAR Txt,Txt2:Text;
    Alphabet:ARRAY[1..27,1..20] OF STRING[20];
    Scr:ARRAY[1..21] OF STRING[20];
    {circular buffer containing a part of the screen}
    Scrpos:Byte;
    {the current position in the circular buffer}

    J,K,Y:Integer;
    Outputmatches,Outputchar:Integer;
    lines:INTEGER;

PROCEDURE Read_Line_Into_Scr; {read the next verticle line into the buffer}
BEGIN
  IF NOT(EOF(Txt)) THEN
  BEGIN
   Readln(Txt,Scr[Scrpos]);
   INC(Scrpos);
   IF Scrpos>21 THEN Scrpos:=1; {let the buffer wrap if it exceeds 21}
  END;
 Dec(lines);
END;

{Procedure decription:
 This procedure compares the 27 given correct characters with a position
 on the screen and record the number of pixels which are similar on the
 screen. For each one of the 27 correct characters it compares another 40
 times for the 20 different kind of ways a verticle line can be ommitted
 or 20 kinds of duplications. Thus in total 27x41 characters are compared
 to a position on the screen. Whichever one fits the best, are selected as
 the character (since it's the greatest possibility to be correct.}

PROCEDURE Scan_Character;
VAR Check_Char,Inloop,Outloop,Looplength,Actionline,Scan_Position:Integer;
    Alphscanpos,J:Integer;
    Task:Integer;{0-leave 1-double 2-ommit}
    O_M,O_C:Integer;
BEGIN
 Outputmatches:=0;
 Outputchar:=1;
 FOR Check_Char:=1 TO 27 DO {loop through all the correct characters}
 BEGIN
  {loop and change to all possible ommitions and duplications of lines 2*20+1}
  FOR Outloop:=0 TO 40 DO
  BEGIN
   CASE Outloop OF {set the changes on the character to be compared}
    0:BEGIN Task:=0;Actionline:=0;Looplength:=20;END;
    1..20:BEGIN Task:=1;Actionline:=Outloop;Looplength:=21;END;
    ELSE BEGIN Task:=2;Actionline:=Outloop-20;Looplength:=19;END;
   END;

   Scan_Position:=Scrpos+1;
   IF Scan_Position>21 THEN Dec(Scan_Position,21); {wrap the buffer}

   Alphscanpos:=1;O_M:=0;O_C:=0;
   FOR Inloop:=1 TO Looplength DO
   BEGIN
    IF Inloop=Actionline THEN
    BEGIN
     IF Task=1 THEN
     BEGIN
      FOR J:=1 TO 20 DO
       IF Alphabet[Check_Char,Alphscanpos][J]=Scr[Scan_Position][J] THEN INC(O_M);
     END ELSE INC(Alphscanpos);
    END ELSE
    BEGIN
     FOR J:=1 TO 20 DO
      IF Alphabet[Check_Char,Alphscanpos][J]=Scr[Scan_Position][J] THEN INC(O_M);
     INC(Alphscanpos);
    END;
    INC(Scan_Position);
    IF Scan_Position>21 THEN Scan_Position:=1;
   END;

   IF Outputmatches<O_M THEN {if the current character has less differences
    then record it as the "closest match character"}
   BEGIN
    Outputchar:=Check_Char;
    Outputmatches:=O_M;
   END;

  END;
 END;
END;

PROCEDURE Output(No:Integer);
BEGIN
 Assign(Txt2,OutputFilename);
 Append(Txt2);
 IF No=1 THEN
 BEGIN
  Write(' ');
  Write(Txt2,' ')
 END ELSE
 BEGIN
  Write(Chr(No+63));
  Write(Txt2,Chr(No+63))
 END;
 Close(Txt2);
END;

PROCEDURE Init;
BEGIN
 Randomize;

 Assign(Txt2,OutputFilename);
 Rewrite(Txt2);
 Close(Txt2);

 Assign(Txt,FontFilename);
 Reset(Txt);
 Readln(txt,j);
 FOR J:=1 TO 27 DO
 BEGIN
  FOR Y:=1 TO 20 DO
  BEGIN
   Readln(Txt,Alphabet[J,Y]);
  END;
 END;
 Close(Txt);

 Assign(Txt,ImageFilename);
 Reset(Txt);
 READLN(txt,lines);

 Scrpos:=1;
 FOR J:=1 TO 21 DO
 BEGIN
  Read_Line_Into_Scr;
 END;
 Scan_Character;
 Output(Outputchar);
 IF NOT(EOF(Txt)) THEN
 FOR J:=1 TO 19 DO {fill the buffer}
   Read_Line_Into_Scr;
END;

VAR O_M,O_C,Gotnum:Integer;
BEGIN
 Clrscr;
 Init;

 {Description: Verticle lines can either be ommitted duplicated or just left
  as they are. The first character wil definitely start on the first verticle
  line of the screen, since there is no character before it which can affect
  the starting verticle line of the character on the screen. If a character
  is then recognised (the character with the highest probability) then
  there is still no certain way to say whether a line was ommitted or
  duplicated. This leaves an uncertainty of where the next character should
  start. This uncertainty will grow with each new character scanned.

  To overcome this problem the program scans all three possible starting
  points of a character. The character with the higest probabality out of
  the possible 3x27x41 character combined positions is then selected.
  The accuracy with this method is surprisingly high.
  }
 IF NOT(EOF(txt)) THEN
 REPEAT
  O_M:=0;O_C:=1;Gotnum:=1;
  FOR J:=1 TO 3 DO {loop all three possible places where the character might
                    start}
  BEGIN
   Scan_Character;
   IF O_M<Outputmatches THEN {check if the character found by scan_character
                              is a better match}
   BEGIN
    O_M:=Outputmatches; {if it is then record the character}
    O_C:=Outputchar;
    Gotnum:=J;
   END;
   Read_Line_Into_Scr; {read another line into the buffer}
  END;
  Output(O_C); {output the end result}
  {depending on where the character was found, scan a number of lines on
   to the next 3 possible character positions}
  FOR J:=1 TO 15+Gotnum DO
  BEGIN
   Read_Line_Into_Scr;
  END
 UNTIL Lines<=0;

 Close(Txt);

END.
```

Stacking containers

```
Uses Crt,StackLib;
Var X,Y,Z,A,B,C:Integer;
    KX,KY,XX,YY,ZZ,MX,MY,MZ:Integer;
    MaxT:Integer;
    Telpa:Array[1..32,1..32,1..32]Of Byte;
    Aug:Array[1..32,1..32]Of Byte;
    Time:Array[1..250]Of Integer;
    Ir:Array[1..250]Of Boolean;
Begin
 XX:=GetX;
 YY:=GetY;
 ZZ:=GetZ;
 For X:=1 To 250 DO Ir[X]:=False;
 C:=GetNextContainer;
 While C<>0 Do
  Begin
   A:=GetNextAction;
   B:=0;
   For X:=1 To XX Do
    For Y:=1 To YY Do B:=B+(ZZ-Aug[X,Y]);
   If(B<ZZ-1)And(A=1)Then RefuseContainer Else
   If A=1 Then
    Begin
     B:=GetNextStorageTime;
     Time[C]:=B;
     Ir[C]:=True;
     MaxT:=-1000;MX:=0;MY:=0;
     For X:=1 To XX Do
      For Y:=1 To YY Do
       If(Aug[X,Y]=0)Or((Time[Telpa[X,Y,Aug[X,Y]]]>MaxT)And(Aug[X,Y]<ZZ))Then
        Begin
         If Aug[X,Y]=0 Then MaxT:=1000 Else
          MaxT:=Time[Telpa[X,Y,Aug[X,Y]]];
         MX:=X;MY:=Y;
        End;
     If MX=0 Then
      Begin
       RefuseContainer;
      End Else
      Begin
       Aug[MX,MY]:=Aug[MX,MY]+1;
       Telpa[MX,MY,Aug[MX,MY]]:=C;
       StoreArrivingContainer(MX,MY);
      End;
    End Else
    BEGIN
     For X:=1 To XX Do
      For Y:=1 To YY Do
       For Z:=1 To ZZ do If Telpa[X,Y,Z]=C Then
        Begin
         MX:=X;MY:=Y;MZ:=Z;
        End;
     While Telpa[MX,MY,Aug[MX,MY]]<>C Do
      Begin
       MaxT:=-1000;KX:=0;KY:=0;
       For X:=1 To XX Do
        For Y:=1 To YY Do
         If(Aug[X,Y]=0)Or(Time[Telpa[X,Y,Aug[X,Y]]]>MaxT)And(Aug[X,Y]<ZZ)Then
          Begin
```

```
           If Aug[X,Y]=0 Then MaxT:=1000 Else
            MaxT:=Time[Telpa[X,Y,Aug[X,Y]]];
           KX:=X;KY:=Y;
          End;
       If KX=0 Then
        Begin
         Halt;
        End Else
        Begin
         Aug[KX,Ky]:=Aug[KX,KY]+1;
         Telpa[Kx,KY,Aug[KX,KY]]:=Telpa[Mx,MY,Aug[MX,MY]];
         Telpa[Mx,MY,Aug[MX,MY]]:=0;
         Aug[MX,MY]:=Aug[MX,MY]-1;
         If MoveContainer(MX,MY,KX,KY)=0 Then
          Begin
           Halt;
          End;
        End;
      End;
     RemoveContainer(MX,MY);
     Telpa[MX,MY,Aug[MX,MY]]:=0;
     Aug[MX,MY]:=Aug[Mx,MY]-1;
     Ir[C]:=False;Time[C]:=0;
    END;
   For X:=1 To 250 Do If Ir[X] then Time[X]:=Time[X]-1;
   C:=GetNextContainer;
  End;
End.
```