Ejercicio 1

{ Este programa lee de la entrada estándar una secuencia de dígitos terminada con un '.' y muestra por la salida estándar si el número natural que representa dicha secuencia es divisible por 5 y por 11, aplicando las siguientes reglas de divisibilidad:
** Un número es divisible por 5 si su dígito menos significativo es 0 o 5.*
** Un número es divisible por 11 si la suma de las cifras en posición par menos la suma de las cifras en posición impar es cero o múltiplo de 11. Para ello, basta con ir sumando y restando los valores de las cifras a medida que son leídas, ajustando el resultado a un valor entre 0 y 10 (sumando o restando 11, es decir, módulo 11). Si la suma final es cero, el número es múltiplo de 11. }*

```
PROGRAM Divisibilidad_11y5 (input, output);
const  chFinSec = '.';
var c, ultimo : char;
    sumaMod_11 : integer;
    posPar : boolean;

begin
  write('introduzca un numero natural: ');
  sumaMod_11 := 0;   posPar := false;   read(c);
  while c <> chFinSec do begin
    if posPar then sumaMod_11 := sumaMod_11 + ord(c) - ord('0')
              else sumaMod_11 := sumaMod_11 - ord(c) - ord('0');
    posPar := not posPar;
    if sumaMod_11 < 0 then sumaMod_11 := sumaMod_11 + 11
    else if sumaMod_11 >= 11 then sumaMod_11 := sumaMod_11 - 11;
    ultimo := c;   write(c);   read(c)
  end;
  writeln;   write('es multiplo de : ');
  if (ultimo = '0') or (ultimo = '5') then write(5:4);
  if sumaMod_11 = 0 then write(11:4);
  writeln
end.
```

Ejercicio 2

{ Este programa lee de la entrada estándar una línea con un mensaje y muestra por la salida estándar el mensaje codificado en MORSE. }

```
PROGRAM codif_Morse_1 (input, output);
type tpCodigo = string[5]; { un código MORSE puede tener hasta 5 caracteres }
     tpTblCod = array [char] of tpCodigo;

procedure cargarCodigo(var tblCodigo: tpTblCod);
type tpCodigoChar = record
                      ch: char;
                      codMorse: tpCodigo
                    end;
var  codigoChar: tpCodigoChar;
     fCod: file of tpCodigoChar;
begin
  assign(fCod, 'codMorse.dat');  reset(fCod);
  { for letra := chr(0) to chr(255) do tblCodigo[letra] := ";} { no es necesario iniciar la tabla pues se supone que no hay errores }
  while not eof(fCod) do begin
    read(fCod, codigoChar);
    tblCodigo[codigoChar.ch] := codigoChar.codMorse
  end;
  close(fCod)
end;

var tblMorse : tpTblCod;
    ch : char;

begin
  cargarCodigo(tblMorse);
  writeln('escriba el texto a codificar:');
  while not eoln do begin
    read(ch);
    if ch = ' ' then write('/')
    else write(tblMorse[ch], '/')
  end;
  readln; writeln('/')
end.
```

Ejercicio 3

```
const DIM_MAX = 100;   OCUPADO = true;   VACIO = false;
type  tpVoxel = boolean;
      tpCoordenada = 1..DIM_MAX;
      tpObjeto = record
                   minX, maxX, minY, maxY, minZ, maxZ: tpCoordenada;
                   vol: array [tpCoordenada, tpCoordenada, tpCoordenada] of tpVoxel
                 end;
```

```
function esEspurio(x, y, z: integer): boolean; {devuelve true sii alguna de las coordenadas no es válida (espurio) }
begin
  esEspurio := (x<1) or (x>DIM_MAX) or (y<1) or (y>DIM_MAX) or (z<1) or (z>DIM_MAX)
end;
```

```
procedure cargarObjeto(var fEsc: text;  var obj: tpObjeto);
{ almacena en obj los vóxeles especificados en el fichero de texto fEsc así como los rangos de coordenadas que delimitan el objeto}
var  x, y, z: integer;    {integer ya que puede haber espurios}
begin
  reset(fEsc);
  for x:=1 to DIM_MAX do
   for y:=1 to DIM_MAX do
    for z:=1 to DIM_MAX do obj.vol[x, y, z] := VACIO;
  repeat readln(fEsc, x, y, z) until not esEspurio(x, y, z); {siempre hay al menos un vóxel válido}
  obj.minX:=x;  obj.maxX:=x;  obj.minY:=y;  obj.maxY:=y;  obj.minZ:=z;  obj.maxZ:=z;
  obj.vol[x,y,z] := OCUPADO; {no es necesario comprobar si estaba vacío, pues no se cuentan }
  while not eof(fEsc) do begin
    readln(fEsc, x, y, z);
    if not esEspurio(x, y, z) then begin
      obj.vol[x,y,z] := OCUPADO; { no es necesario comprobar si estaba vacío, pues no se cuentan }
      if x>obj.maxX then obj.maxX:=x else if x<obj.minX then obj.minX:=x;
      if y>obj.maxY then obj.maxY:=y else if y<obj.minY then obj.minY:=y;
      if z>obj.maxZ then obj.maxZ:=z else if z<obj.minZ then obj.minZ:=z
    end
  end
end;
```

```
function porcentajeOcupacion(var obj: tpObjeto): real;
var  x, y, z: tpCoordenada;
     contador, volPrisma: integer;
begin
  contador:=0;
  for x:=obj.minX to obj.maxX do
   for y:=obj.minY to obj.maxY do
    for z:=obj.minZ to obj.maxZ do if obj.vol[x,y,z]=OCUPADO then contador:=contador+1;
  volPrisma := (obj.maxX-obj.minX+1)*(obj.maxY-obj.minY+1)*(obj.maxZ-obj.minZ+1);
  porcentajeOcupacion := contador * 100 / volPrisma
end;
```