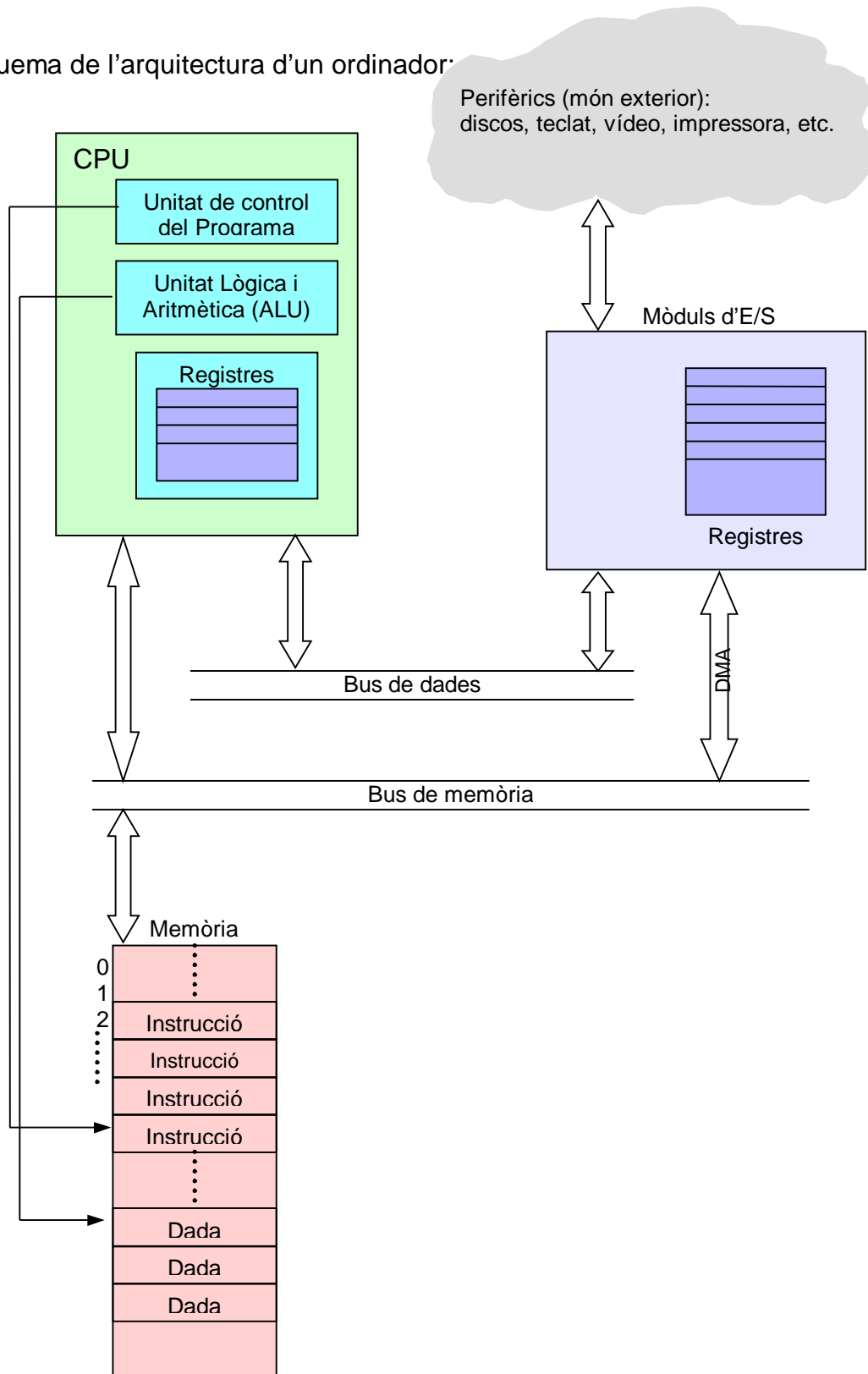


Fonaments d'Informàtica
Solucions de les pràctiques
Curs 2001/02



Pràctica 1. Arquitectura del computador

1.2 Esquema de l'arquitectura d'un ordinador:



1.3 Imagineu-vos que sou venedors/res d'una botiga d'informàtica. Aconselleu als següents cinc clients que arriben a la vostra botiga sobre quin és l'ordinador s'ajusta a les seves necessitats, és a dir, quins components i accessoris necessitaran, com han de ser aquests i el perquè.

- Client 1: Na Júlia és una enginyera. A la feina necessita un ordinador per fer simulacions de dinàmiques de fluids (càlculs molt costosos i moltíssimes dades). La resta d'ordinadors de la feina accediran constantment a aquest ordinador mitjançant la seva xarxa local o intranet per consultar els resultats de les simulacions.
- Client 2: En Xavi és un noi que ja té un ordinador. L'utilitza sobretot per jugar: és un apassionat dels videojocs. El problema és que els últims jocs que ha comprat ja van una mica lents en el seu ordinador. Quins components del seu ordinador li cal canviar per millorar el rendiment d'aquest a l'hora de jugar amb els jocs? A més a més, en Xavi i la seva germana Eulàlia estan aprenent solfeig i volen formar un grup. Volen utilitzar l'ordinador com a assistent a l'hora de compondre música.
- Client 3: En Joan treballa a una oficina. Vol un ordinador per a poder escriure a casa informes i fer feina del treball a casa. Com és l'ordinador que li vendríeu?

Component	Client 1	Client 2	Client 3
microprocessador	↑ ràpid (càlculs i atendre peticions)	↑ ràpid (càlculs)	-
memòria RAM	↑ ràpida (càlculs) ↑ gran (dades)	↑ ràpida (càlculs) ↑ gran (dades textures)	-
ratolí	-	-	-
teclat	-	-	-
joystick	-	-	-
disquetera	-	-	-
disc dur	↑ gran (dades) ↑ ràpid (atendre peticions)	-	-
CD-ROM	-	pot ser (dades del joc en CD)	-
DVD	-	-	-
gravadora CD-ROM	- (per còpies de seguretat)	-	-
escàner	-	-	-
impressora	-	-	-
monitor	-	↑ ràpid (freqüència)	-
mòdem	?	?	?
tarja de xarxa	↑ bona (atendre peticions)	-	-
tarja de vídeo	-	↑ ràpida (càlculs)	-
tarja de so	-	↑ (qualitat so/pistes)	-
altaveus	-	↑ (qualitat so)	-
caixa	-	-	portàtil
s.a.i.	↑ (talls de llum)	-	-

Pràctica 2. Codificació

2.1 Canvis de base.

Completar la següent taula:

Binari	Hexadecimal	Decimal
10000000.11	80.C	128.75
10011101.11001	9D.C8	157.78125
10111110.10100111	BE.A7	190.65234375

2.2 Codificació de nombres enters amb signe.

a) Representar els nombres 223 i -223 en binari en els diferents codis per a la representació d'enters amb signe.

		Binari
Binari natural	223	11011111
	-223	No es pot representar
Signe i magnitud	223	011011111
	-223	111011111
Complement a 2	223	011011111
	-223	100100001
Excés (256)	223	111011111
	-223	000100001

b) Calcular el valor decimal del nombre binari $10100111_{(2)}$ representat en els diferents codis.

		Decimal
Binari Natural	10100111 ₍₂₎	167
Signe i Magnitud	10100111 ₍₂₎	-39
Complement a 2	10100111 ₍₂₎	-89
Excés (128)	10100111 ₍₂₎	39

2.3 Aritmètica binària.

Realitzar les següents operacions en 8 bits i complement a 2, donant el valor del resultat en decimal.

Sumar

$$26 + 24 = 50$$

$$-15 + 82 = 67$$

$$84 + 69 = 103$$

$$-46 + (-10) = -56$$

$$\begin{array}{r}
 + 26 \\
 + 24 \\
 + 50 \\
 \hline
 \end{array}
 \begin{array}{|c|c|c|c|c|c|c|c|}
 \triple{0}{0}{0}{1}{1}{0}{1}{0} \\
 \triple{0}{0}{0}{1}{1}{0}{0}{0} \\
 \triple{0}{0}{1}{1}{0}{0}{1}{0} \\
 \triple{0}{0}{1}{1}{0}{0}{1}{0} \\
 \end{array}$$

$$\begin{array}{r}
 - 15 \\
 + 82 \\
 + 67 \\
 \hline
 \end{array}
 \begin{array}{|c|c|c|c|c|c|c|c|}
 \triple{1}{1}{1}{1}{0}{0}{0}{1} \\
 \triple{0}{1}{0}{1}{0}{0}{1}{0} \\
 \triple{1}{0}{1}{0}{0}{0}{1}{1} \\
 \triple{0}{1}{0}{0}{0}{0}{1}{1} \\
 \end{array}$$

$$\begin{array}{r}
 + 84 \\
 + 69 \\
 - 103 \\
 \hline
 \end{array}
 \begin{array}{|c|c|c|c|c|c|c|c|}
 \triple{0}{1}{0}{1}{0}{1}{0}{0} \\
 \triple{0}{1}{0}{0}{0}{1}{0}{1} \\
 \triple{1}{0}{0}{1}{1}{0}{0}{1} \\
 \triple{0}{1}{1}{0}{0}{1}{1}{1} \\
 \end{array}$$

OVERFLOW !

$$\begin{array}{r}
 - 46 \\
 - 10 \\
 - 56 \\
 \hline
 \end{array}
 \begin{array}{|c|c|c|c|c|c|c|c|}
 \triple{1}{1}{0}{1}{0}{0}{1}{0} \\
 \triple{1}{1}{1}{1}{0}{1}{1}{0} \\
 \triple{1}{1}{0}{0}{1}{0}{0}{0} \\
 \triple{0}{0}{1}{1}{1}{0}{0}{0} \\
 \end{array}$$

2.4 Codificació de nombres reals

Representar 123.12 segons norma IEEE 754 en simple precisió. Utilitzar truncat o arrodoniment si fos necessari.

$$123.12_{(10)} = 1111011.000111101011100001_{(2)} = 1.111011000111101011100001_{(2)} \times 2^6$$

- 123.12 és positiu, per tant el signe és representa amb 0.
- En excés 127, l'exponent 6 es representa com $6+127 = 133_{(10)} = 10000101_{(2)}$
- La mantissa és 111011000111101011100001, que té 24 bits quan hauria de tenir 23 bits. Truncant la mantissa a 23 bits obtenim la nova mantissa 11101100011110101110000 i arrodonint a 23 bits obtenim la nova mantissa 11101100011110101110001. Treballarem amb el valor arrodonit, donat que amb ell obtenim una precisió més gran.

0	1	0	0	0	0	1	0	1	1	1	1	0	1	1	0	0	0	1	1	1	1	0	1	0	1	1	1	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

2.5 Memòria i precisió

En un microprocessador de 16 bits desitgem executar un programa que ocupa 20000 posicions de memòria precisant a més: 10000 dades enteres (16 bits), 5000 dades reals de simple precisió (32 bits), 2000 dades reals de doble precisió (64 bits) i 1000 alarmes binàries (1 bit). Indicar la capacitat mínima de la memòria necessària en posicions de memòria, bits, bytes i quilobytes.

Suposant que una posició de memòria té capacitat per emmagatzemar 16 bits.

- 20000 posicions de memòria ocupa el codi del programa.
- $10000 \text{ dades enteres} \times 16 \text{ bits/dada entera} \times 1 \text{ posició}/16 \text{ bits} = 10000$ posicions
- $5000 \text{ dades reals de simple precisió} \times 32 \text{ bits/dada real} \times 1 \text{ posició}/16 \text{ bits} = 10000$ posicions
- $2000 \text{ dades reals de doble precisió} \times 64 \text{ bits/dada real} \times 1 \text{ posició}/16 \text{ bits} = 8000$ posicions
- $1000 \text{ alarmes binàries} \times 1 \text{ bits/alarma} \times 1 \text{ posició}/16 \text{ bits} = 62.5 = 63$ posicions

La capacitat mínima de la memòria necessària és $20000 + 10000 + 10000 + 8000 + 63 = \mathbf{48063 \text{ posicions de memòria}}$.

$48063 \text{ posicions de memòria} \times 16 \text{ bits/posició} = 769008 \text{ bits}$

$769008 \text{ bits} \times 1 \text{ byte}/8 \text{ bits} = 96126 \text{ bytes}$

$96126 \text{ bytes} \times 1 \text{ Kbyte}/1024 \text{ bytes} = 93.87 \text{ Kbytes}$

2.6 Codi ASCII. Codi detector d'errors. Codis de Paritat.

Volem enviar per una línia telefònica una informació en paquets de 8+1 bits. Aplicar un codi de paritat parella per enviar la cadena de caràcters "Alumne" (sense les cometes) en codi ASCII de 8 bits. Discutir la possibilitat de detectar transmissions errònies.

Aplicar també un codi de paritat senar a la mateixa cadena ASCII.

Caràcter	Codi ASCII	Binari 8 bits	Bit paritat parella	Bit paritat senar
A	065 ₍₁₀₎	01000001 ₍₂₎	0	1
l	108 ₍₁₀₎	01101100 ₍₂₎	0	1
u	117 ₍₁₀₎	01110101 ₍₂₎	1	0
m	109 ₍₁₀₎	01101101 ₍₂₎	1	0
n	110 ₍₁₀₎	01101110 ₍₂₎	1	0
e	101 ₍₁₀₎	01100101 ₍₂₎	0	1

Amb bit de paritat parella la cadena enviada és:

01000001 0 01101100 0 01110101 1 01101101 1 01101110 1 01100101 0

Amb bit de paritat senar la cadena enviada és:

01000001	1	01101100	1	01110101	0	01101101	0	01101110	0	01100101	1
----------	---	----------	---	----------	---	----------	---	----------	---	----------	---

Pràctica 3. Entorn Turbo Pascal

3.1 Suma d'enters

El resultat de les sumes hauria de ser:

```
200 + 350 = 550
250 - 300 = -50
20000 + 30000 = -15536
40000 - 10 = -25546
```

El que ha passat a les dues últimes sumes és que s'ha produït un error de desbordament (overflow). Els nombres de tipus `integer` en Turbo Pascal es codifiquen en binari seguint un sistema de representació de 16 bits i complement a dos. Això fa que el rang de representació dels enters vagi del -32768 al 32767 . Els resultats correctes de la tercera suma, 50000 , i de la quarta suma, 39990 , estan per sobre del nombre enter més gran que podem representar, 32767 .

Si voleu un rang de representació més gran a l'hora de treballar amb nombres enters, feu servir el tipus `longint`. Trobareu més informació d'ell a l'ajuda de Turbo Pascal (tecles CTRL+F1).

3.2 Suma de reals

El resultat de les sumes hauria de ser:

```
250.30 + 300.50 = 5.5080000000E+02
1E12 + 1.0 = 1.0000000000E+12
3333333333333333.333 + 6.667 = 3.3333333333E+14
987654321004 + -987654321002 = 2.0000000000E+00
123456789876543 + 0 = 1.2345678988E+14
```

El tipus `real` del Turbo Pascal té un rang de representació de $2.9e^{-39}$ a $1.7e^{38}$, amb 11-12 xifres significatives. És per això que hi ha una pèrdua de precisió a les sumes segona (13 xifres), tercera (18 xifres) i cinquena (15 xifres). A aquesta última suma podeu observar com el Pascal arrodoneix el nombre real.

Si voleu més precisió a l'hora de treballar amb nombres reals a Turbo Pascal, feu servir els tipus `double` i `extended`. Trobareu més informació sobre ells a l'ajuda de Turbo Pascal (tecles CTRL+F1).

Per cert, sabíeu que el tipus real de simple i doble precisió definits per l'estàndard IEEE 754 que vàrem veure a codificació, a Pascal són el tipus `SINGLE` i `DOUBLE`, respectivament. El tipus `REAL` de 6 bytes és únic del Pascal i generalment incompatible amb altres llenguatges. Normalment l'arquitectura interna dels microprocessadors està optimitzada per treballar amb nombres reals codificats per aquest estàndard de la IEEE. Per motius d'eficiència i compatibilitat és millor treballar amb el tipus `DOUBLE` i, de fet, a Delphi 4 el tipus `REAL` ja és un sobrenom del tipus `DOUBLE`.

Pràctica 4. Seqüencials

4.1 Observeu el següent programa en Pascal, que implementa el càlcul de l'àrea i el perímetre d'un cercle, donat un radi r .

$$\text{Equació de l'àrea d'un cercle: } A = \pi * r^2$$

$$\text{Equació del perímetre d'un cercle: } P = 2 * \pi * r$$

Responeu les següents qüestions:

- Quines variables són d'entrada, quines auxiliars i quines de sortida? **radi és la variable d'entrada, i perímetre i area són les variables de sortida. No hi ha variables auxiliars.**
- Com és que podem fer servir PI, si no ha estat declarat? **És una primitiva del sistema.**
- Per què posem un missatge (WRITE) davant l'entrada de dades (READ)? **Per informar l'usuari que ha d'introduir dades. És obligatori fer-ho així? No. Què passaria si no? Que l'ordinador es quedaria a l'espera de dades per teclat, però l'usuari no rebria cap avís i no sabria que les ha d'introduir.**

Programa en Pascal

```
PROGRAM A4_1;
VAR radi, area, perimetre : REAL;
BEGIN
  (* Demanem el radi *)
  WRITE('Quin és el radi ? ');
  READLN(radi);

  (* Calculem l'àrea i el perímetre *)
  perimetre := 2 * PI * radi;
  area := PI * radi * radi;

  (* Visualitzem els resultats *)
  WRITELN('El perímetre és ', perimetre:4:2);
  WRITELN('L'àrea és ', area:4:2);
END.
```

4.2 Dissenyau un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal, que permetin trobar el punt mig de dos punts de l'espai bidimensional, segons la fórmula:

$$\text{Siguin els punts } \vec{a} = (a_x, a_y) \text{ i } \vec{b} = (b_x, b_y) \text{ llavors } \vec{m} = \vec{a} + \vec{b} = \left(\frac{a_x + b_x}{2}, \frac{a_y + b_y}{2} \right)$$

Programa en Pascal

```
PROGRAM Sequencial;

VAR ax, ay, bx, by, mx, my : REAL;

BEGIN
  (* Demanem el primer punt *)
  WRITE('Coordenades del primer punt separades per espais: ');
  READLN(ax, ay);

  (* Demanem el segon punt *)
  WRITE('Coordenades del segon punt separades per espais: ');
  READLN(bx, by);

  (* Calculem el punt mig *)
  mx := (ax + bx)/2;
  my := (ay + by)/2;

  (* Mostrem el resultat *)
  WRITELN('El punt mig és ( ', mx:4:2, ', ', my:4:2, ' )');
END.
```

4.3 Dissenyem un algorisme en pseudocodi i realitzem el corresponent programa en Pascal que, donat un nombre enter que designa un període de temps expressat en segons, retorni l'equivalent en dies, hores, minuts i segons.

Per exemple: 24000 segons seran 0 dies, 6 hores, 40 minuts i 0 segons.

Per exemple: 7400 segons seran 0 dies, 2 hores, 3 minuts i 20 segons.

Programa en Pascal, solució 1

```
PROGRAM A4_3_a;
VAR temps : LONGINT;
    segons, minuts, hores, dies : INTEGER;
BEGIN
  (* Demanem els segons *)
  WRITE('Introdueix un període de temps expressat en segons : ');
  READLN(temps);

  (* Calculem els segons i el temps que resta en minuts *)
  segons := temps MOD 60;
  temps := temps DIV 60;

  (* Calculem els minuts i el temps que resta en hores *)
  minuts := temps MOD 60;
  temps := temps DIV 60;

  (* Calculem les hores i el temps que resta en dies *)
  hores := temps MOD 24;
  dies := temps DIV 24;

  (* Visualitzem els resultats *)
  WRITELN('Això són ', dies, ' dies, ', hores, ' hores, ', minuts,
    ' minuts i ', segons, ' segons.');
```

```
END.
```

Programa en Pascal, solució 2

```
PROGRAM A4_3_b;
VAR temps : LONGINT;
BEGIN
  (* Demanem el temps en segons *)
  WRITE('Introdueix un període de temps expressat en segons : ');
  READLN(temps);

  (* Imprimim els dies i calculem el temps que resta en segons *)
  WRITE('Això són : ', temps DIV (24*60*60) , ' dies, ');
  temps := temps MOD (24*60*60);

  (* Imprimim les hores i calculem el temps que resta en segons *)
  WRITE(temps DIV (60*60) , ' hores, ');
  temps := temps MOD (60*60);

  (* Imprimim els minuts i imprimim el temps que resta en segons *)
  WRITE(temps DIV 60 , ' minuts i ');
  WRITELN(temps MOD 60 , ' segons.');
```

END.

Pràctica 5. Estructures alternatives

5.1 Siguin A , B i C tres variables enteres que representen les vendes de tres productes A , B i C , respectivament. Utilitzant aquestes variables, escriviu les expressions que representin les següents afirmacions:

a) Les vendes del producte A són les més elevades.

$(A > B) \text{ AND } (A > C)$

b) Cap producte té unes vendes inferiors a 200.

$\text{NOT } ((A < 200) \text{ OR } (B < 200) \text{ OR } (C < 200))$

$(A \geq 200) \text{ AND } (B \geq 200) \text{ AND } (C \geq 200)$

c) Algun producte té unes vendes superiors a 400.

$(A > 400) \text{ OR } (B > 400) \text{ OR } (C > 400)$

d) La mitjana de vendes és superior a 500.

$(A + B + C) / 3 > 500$

e) El producte B no és el més venut.

$(B < A) \text{ OR } (B < C)$

$\text{NOT } ((B > A) \text{ AND } (B > C))$

f) El total de vendes està entre 500 i 1000.

$(500 < (A + B + C)) \text{ AND } ((A + B + C) < 1000)$

L'ús de parèntesis és necessari degut a l'ordre de precedència d'operadors. Per exemple, si escrivíssim $A > B \text{ AND } A > C$ el compilador de Pascal intentaria avaluar l'expressió $A > (B \text{ AND } A) > C$, que donaria error.

5.2 Donada una variable c de tipus caràcter, trobeu les expressions que valguin cert si i només si:

a) c és una vocal.

$(c = 'a') \text{ OR } (c = 'e') \text{ OR } (c = 'i') \text{ OR } (c = 'o') \text{ OR } (c = 'u') \text{ OR}$

$(c = 'A') \text{ OR } (c = 'E') \text{ OR } (c = 'I') \text{ OR } (c = 'O') \text{ OR } (c = 'U')$

$c \text{ in } ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']$

b) c és una lletra minúscula.

$(c \geq 'a') \text{ AND } (c \leq 'z')$

$c \text{ in } ['a'..'z']$

c) *c* és un símbol de l'alfabet.

```
((c >= 'a') AND (c <= 'z')) OR ((c >= 'A') AND (c <= 'Z'))  
c in ['a'..'z' , 'A'..'Z']
```

A les expressions de pertinença a un conjunt (operador `IN`) hem de tenir cura de que els elements siguin d'un tipus numerable (enter o caràcter), el nombre d'elements del conjunt no superi els 256, i que els seus valors estiguin entre 0 i 255.

5.3 Observeu el següent programa en Pascal que, introduïts tres nombres qualsevol pel teclat, calcula el mínim i el màxim dels tres nombres i els mostra per pantalla.

Responen les següents qüestions:

- Per què s'utilitzen estructures alternatives niuades? **Per aprofitar la informació rebuda en avaluar una expressió lògica, i així no fer preguntes redundants, augmentant així l'eficiència del codi. Al `IF` en negreta, si *c* és el màxim ja no cal preguntar si és el mínim.**
- Fa servir comparacions repetides o innecessàries? **No.** Es podria fer d'una altra manera amb menys comparacions? **Crec que no.**

Programa en Pascal

```
PROGRAM A5_3;  
  
VAR min, max : REAL;          (* Aquí guardarem el mínim i el màxim *)  
    a, b, c  : REAL;          (* Aquí guardarem el valors introduït *)  
  
BEGIN  
    WRITE('Introdueix el primer valor ');  
    READLN(a);  
    min := a;  
    max := a;  
  
    WRITE('Introdueix el segon valor ');  
    READLN(b);  
    IF b > max THEN  
        max := b  
    ELSE  
        min := b;  
  
    WRITE('Introdueix el tercer valor ');  
    READLN(c);  
    IF c > max THEN  
        max := c  
    ELSE  
        IF c < min THEN  
            min := c;  
  
    WRITELN('El mínim és ', min);
```

```
WRITELN('El màxim és ', max);  
END.
```

5.4 Observeu el següent programa en Pascal que, a partir del número del dia de la setmana introduït (de 1 a 7), si aquest és laboral escriu el nom del dia corresponent per la pantalla, i si no escriu festiu.

Responen les següents qüestions:

- Quan es pot emprar l'alternativa múltiple a Pascal i quan no? **Es pot emprar quan l'expressió i els valors a comparar són d'un tipus numerable, és a dir, enter o caràcter.**
- Es podria escriure aquest programa amb estructures alternatives IF niuades, enlloc de l'alternativa múltiple CASE? **Sí, tota estructura CASE es pot substituir per estructures IF niuades. Al programa d'exemple quedaria:**

```
IF dia = 1 THEN  
  WRITELN('dilluns')  
ELSE  
  IF dia = 2 THEN  
    WRITELN('dimarts')  
  ELSE  
    IF dia = 3 THEN  
      WRITELN('dimecres')  
    ELSE  
      IF dia = 4 THEN  
        WRITELN('dijous')  
      ELSE  
        IF dia = 5 THEN  
          WRITELN('divendres')  
        ELSE  
          IF (dia = 6) OR (dia = 7) THEN  
            WRITELN('festiu')  
          ELSE  
            WRITELN('incorrecte');  
          END IF  
        END IF  
      END IF  
    END IF  
END IF
```

- Quines avantatges té llavors el CASE sobre l'IF? **Millor llegibilitat i més facilitat per afegir noves comparacions.**

Programa en Pascal

```
PROGRAM A5_4;  
  
VAR dia : 1..7;  
  
BEGIN  
  WRITE('Introdueix un dia de la setmana (entre 1 i 7) : ');  
  READLN(dia);  
  
  WRITE(' El dia és ... ');  
  
  CASE dia OF  
    1: WRITELN('dilluns');  
    2: WRITELN('dimarts');  
    3: WRITELN('dimecres');  
    4: WRITELN('dijous');  
    5: WRITELN('divendres');
```

```
6,7: WRITELN('festiu');  
ELSE  
    WRITELN('incorrecte');  
END;  
END.
```

5.5 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal tal que, introduïts dos nombres qualsevol a i b per teclat, trobin la solució a l'equació de primer grau:

$$a x + b = 0$$

Tingueu en compte que hi ha tres possibles solucions:

- Quan $a \neq 0$ existeix la solució única $x = -b/a$.
- Quan $a = 0$ i $b \neq 0$ no existeix solució.
- Quan $a = 0$ i $b = 0$ existeixen infinites solucions.

Programa en Pascal

```
PROGRAM A5_5;  
  
VAR a, b : REAL;          (* coeficients de l'equació *)  
  
BEGIN  
    WRITELN('Resolució d'equacions de primer grau: a x + b = 0');  
  
    WRITE('Introdueix el primer coeficient: a = ');  
    READLN(a);  
  
    WRITE('Introdueix el segon coeficient: b = ');  
    READLN(b);  
  
    IF a <> 0 THEN  
        WRITELN('La solució de l'equació és x = ', -b/a)  
    ELSE  
        IF b = 0 THEN  
            WRITELN('L'equació té infinites solucions.')  
        ELSE  
            WRITELN('L'equació no en té cap solució.');    END.
```

5.6 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal tal que, introduïda la mida d'un cargol pel teclat mostrin per pantalla el text corresponent a la mida, segons la següent taula:

De 1 cm. (inclòs) fins a 3 cm. (no inclòs)	Petit
De 3 cm. (inclòs) fins a 5 cm. (no inclòs)	Mitjà
De 5 cm. (inclòs) fins a 6.5 cm. (no inclòs)	Gran
De 6.5 cm. (inclòs) fins a 8.5 cm. (no inclòs)	Molt gran

Programa en Pascal

```
PROGRAM A5_6;

VAR mida : REAL;

BEGIN
  WRITE('Introdueix la mida del cargol : ');
  READLN(mida);

  IF (mida < 1) OR (mida >= 8.5) THEN
    WRITELN('Mida incorrecte')
  ELSE
    IF mida < 3 THEN
      WRITELN('Petit')
    ELSE
      IF mida < 5 THEN
        WRITELN('Mitjà')
      ELSE
        IF mida < 6.5 THEN
          WRITELN('Gran')
        ELSE
          WRITELN('Molt gran');
    END.
END.
```

5.7 Amplieu l'exercici 4.2 per tal que, a més a més de trobar el punt mig dels dos punts de l'espai bidimensional, escrigui per pantalla a quin quadrant del pla pertany aquest punt resultant.

Programa en Pascal

```
PROGRAM A5_7;

VAR ax, ay, bx, by, mx, my : REAL;

BEGIN
  (* Demanem el primer punt *)
  WRITE('Coordenades del primer punt separades per espais: ');
  READLN(ax, ay);

  (* Demanem el segon punt *)
  WRITE('Coordenades del primer punt separades per espais: ');
  READLN(bx, by);

  (* Calculem el punt mig *)
  mx := (ax + bx)/2;
  my := (ay + by)/2;

  (* Mostrem els resultats *)
  WRITELN('El punt mig és (' , mx:4:2, ', ', my:4:2, ')');

  (* Comprovem i mostrem el quadrant del punt resultat *)
  IF mx > 0 THEN
    IF my > 0 THEN
      WRITELN('1er. quadrant')
    ELSE
      IF my < 0 THEN
        WRITELN('4rt. quadrant')
      ELSE
        WRITELN('Sobre l''eix X')
    END.
  END.
END.
```

```

ELSE
  IF mx < 0 THEN
    IF my > 0 THEN
      WRITELN('2on. quadrant')
    ELSE
      IF my < 0 THEN
        WRITELN('3er. Quadrant')
      ELSE
        WRITELN('Sobre l'eix X')
    ELSE
      IF my <> 0 THEN
        WRITELN('Sobre l'eix Y')
      ELSE
        WRITELN('Origen de coordenades');
END.

```

5.8 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que ens permetin resoldre l'equació de segon grau $a x^2 + b x + c = 0$. La formula matemàtica que resol aquesta equació és la següent:

$$x = \frac{b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2}$$

Cal tenir en compte els casos 'especials' en la seva resolució:

- L'equació és de primer grau ($a = 0$), però es pot calcular el resultat fent servir l'algorisme de l'exercici 5.5.
- Les arrels són imaginàries ($b^2 - 4ac < 0$), però es pot mostrar el resultat separant la part real de la imaginària.

En Pascal, per calcular l'arrel quadrada podem utilitzar SQRT(x) i per elevar al quadrat SQR(x).

Programa en Pascal

```

PROGRAM A5_8;

VAR a , b , c      : REAL;
    discriminant  : REAL;

BEGIN
  WRITELN('Programa per resoldre equacions de segon grau');
  WRITELN('=====');

  (* Demanem els coeficients de l'equació *)
  WRITE('Introdueixi el coeficient del terme de grau 2 : ');
  READLN(a);
  WRITE('Introdueixi el coeficient del terme de grau 1 : ');
  READLN(b);
  WRITE('Introdueixi el coeficient del terme de grau 0 : ');
  READLN(c);

  (* Comprovem que l'equació realment sigui de segon grau *)
  IF a = 0 THEN
    WRITELN('L'equació no és de segon grau.')
    (* Una altre opció era resoldre l'equació de 1er grau *)
  ELSE

```

```

BEGIN
discriminant := b*b - 4*a*c;
IF discriminant = 0 THEN
  BEGIN
  WRITELN('Solució única:');
  WRITELN('x = ', -b/(2*a));
  END
ELSE
  IF discriminant > 0 THEN
  BEGIN
  WRITELN('Dues solucions reals:');
  WRITELN('x = ', (-b + SQRT(discriminant))/(2*a));
  WRITELN('x = ', (-b - SQRT(discriminant))/(2*a));
  END
  ELSE
  BEGIN
  WRITELN('Dues solucions imaginàries:');
  WRITELN('x = ', -b/(2*a), ' ± ', SQRT(-discriminant)/(2*a),
'i');
  END
  END;
END.

```

5.9 Dissenyau un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que ens permetin calcular l'arcsinus d'un determinat valor x a partir de la funció arctangent. Cal tenir en compte que la funció $\text{ArcSin}(x)$ no existeix en el Pascal i el que farem és utilitzar l'equivalència trigonomètrica. Els valors 1 i -1 reben un tracte especial. Al final, el valor resultant en radians s'haurà de passar a graus.

$$\arcsin(x) = \arctan\left(\frac{x}{\sqrt{1-x^2}}\right) \quad \forall x \in]-1, 1[$$

$$\arcsin(-1) = -\frac{\pi}{2}$$

$$\arcsin(1) = \frac{\pi}{2}$$

Programa en Pascal

```

PROGRAM A5_9;
VAR x , arcsin : REAL;
BEGIN
  WRITE('Introdueix un nombre entre -1 i 1 i calcularé l'arcsinus: ');
  READLN(x);
  IF (x < -1) OR (x > 1) THEN
    WRITELN('Valor incorrecte')
  ELSE
    BEGIN
      IF x = -1 THEN
        arcsin := -PI/2
      ELSE
        IF x = 1 THEN
          arcsin := PI/2
        ELSE

```

```
        arcsin := arctan(x/sqrt(1-sqr(x)));
arcsin := arcsin *180 / PI;
WRITELN('L'arcsinus de ', x:5:3, ' és ', arcsin:5:3, ' graus');
END;
END.
```

5.10 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que, donada una data (dia, mes i any), determini si la data correspon a un valor vàlid. Les dates seran tres dades de tipus enter que correspondran a un dia, un mes i un any (dd, mm, aa).

S'ha de tenir present el valor dels dies en funció dels mesos i dels anys. És a dir:

- Els mesos 1, 3, 5, 7, 8, 10 i 12 tenen 31 dies.
- Els mesos 4, 6, 9 i 11 tenen 30 dies.
- El mes 2 te 28 dies, excepte quan l'any és divisible per 4, que té 29 dies.

Programa en Pascal

```
PROGRAM A5_10;

VAR dia, mes, any : INTEGER;

BEGIN
  WRITE('Introdueixi una data (dia, mes i any) : ');
  READLN(dia, mes, any);

  CASE mes OF

    1,3,5,7,8,10,12 : IF (dia >= 1) AND (dia <= 31) THEN
                      WRITELN('Data correcta')
                    ELSE
                      WRITELN('Data incorrecta');

    4,6,9,11       : IF (dia >= 1) AND (dia <= 30) THEN
                      WRITELN('Data correcta')
                    ELSE
                      WRITELN('Data incorrecta');

    2              : IF (dia >= 1) AND ((dia <= 28) OR
                      ((dia = 29) AND ((any MOD 4) = 0))) THEN
                      WRITELN('Data correcta')
                    ELSE
                      WRITELN('Data incorrecta');

  ELSE WRITELN('Data incorrecta');

END;
END.
```

5.11 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que ens permeti escollir entre les següents opcions:

1. Calcular el sinus d'un nombre a.

2. Calcular el logaritme neperià d'un nombre a .
3. Calcular el logaritme en base b d'un nombre a .

Cal observar que no existeix en Pascal una funció que permeti directament el logaritme en una base qualsevol. En el seu lloc podem fer servir la següent fórmula: $\log_b a = \ln a / \ln b$

Programa en Pascal

```
PROGRAM A5_11;

VAR opcio : INTEGER;
    a , b : REAL;

BEGIN
    WRITELN;
    WRITELN('-----');
    WRITELN('1. Sinus');
    WRITELN('2. Logaritme neperià');
    WRITELN('3. Logaritme en una base qualsevol');
    WRITELN('-----');

    WRITE('Quina opció vol ? ');
    READLN(opcio);
    WRITELN;

    CASE opcio OF

        1 : BEGIN
            WRITE('Introdueix un nombre en graus ');
            READLN(a);
            WRITELN('El sinus de ', a:4:2, ' és ', SIN(a*PI/180):5:3);
            END;

        2 : BEGIN
            WRITE('Introdueix un nombre ');
            READLN(a);
            WRITELN('El logaritme neperià de ', a:4:2, ' és ',
                LN(a):5:3);
            END;

        3 : BEGIN
            WRITE('Introdueix un nombre ');
            READLN(a);
            WRITE('Introdueix la base ');
            READLN(b);
            WRITELN('El logaritme en base ', b:4:2, ' de ', a:4:2, ' és ',
                LN(a)/LN(b):5:3);
            END;

    ELSE
        WRITELN('Opció incorrecta');

    END;

END.
```

Pràctica 6. Estructures iteratives

6.1 Observeu el següent programa en Pascal que calcula la mitjana entre una sèrie de valors que l'usuari introduirà per teclat fins que no entri el valor 0.

Responen les següents qüestions:

- Què fan les instruccions del tipus `variable := variable + expressió aritmètica`? **Acumular el valor de l'expressió aritmètica al valor que ja tenia la variable.**
- Com es podria haver realitzat amb una estructura iterativa de condició inicial (WHILE)?

```
(* Inicialitzem les variables *)
sumatori:=0;
terme:=0;

(* Llegim dades i les processem mentre no introdueixin un zero *)
WRITE('Introdueix una dada (0 per acabar) : ');
READLN(x);
WHILE x <> 0 DO
  BEGIN
    sumatori := sumatori + x;
    terme := terme + 1;
    WRITE('Introdueix una dada (0 per acabar) : ');
    READLN(x);
  END;

(* Calculem la mitjana (ara no hem comptat un terme de més). *)
mitjana := sumatori / terme;
```

- Com es podria haver realitzat amb una estructura iterativa repetitiva (FOR)? **Amb l'estructura iterativa repetitiva (FOR) no es podria realitzar, degut a què quan executem el programa no sabem el nombre de termes que introduirà l'usuari.**
- En quin cas especial l'algorisme i el programa no funcionen? **En el cas que l'usuari no introdueixi cap valor. És a dir, que el primer valor ja sigui el 0.** Com ho podríem solucionar? **Afegint una estructura alternativa a l'hora de calcular i visualitzar la mitjana.**

```
(* Calculem la mitjana (comprovant que tinguem termes). *)
IF (terme - 1) = 0 THEN
  BEGIN
    WRITELN('No puc calcular la mitjana. No ha introduït cap dada!');
  END
ELSE
  BEGIN
    mitjana := sumatori / (terme-1);
    WRITELN('La mitjana dels elements entrats és ', mitjana:8:2);
  END;
```

Programa en Pascal

```

PROGRAM A6_1;

VAR x, mitjana, sumatori: REAL;
    terme: INTEGER;

BEGIN
    WRITELN('Càlcul de la mitjana d''una sèrie de dades');
    WRITELN('-----');

    (* Inicialitzem les variables *)
    sumatori:=0;
    terme:=0;

    (* Llegim dades i les processem fins que introdueixin un zero *)
    REPEAT
        WRITE('Introdueix una dada (0 per acabar) : ');
        READLN(x);
        sumatori := sumatori + x;
        terme := terme + 1;
    UNTIL x=0;

    (* Com hem contat el zero com a terme, restem 1 al nombre *)
    (* de termes a l'hora de calcular la mitjana. *)
    mitjana := sumatori / (terme-1);

    (* Donem els resultats *)
    WRITELN(La mitjana dels elements entrats és ', mitjana:8:2);
END.

```

6.2 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que presentin els elements de la següent sèrie i també la seva suma:

$$Sèrie = \sum_{i=1}^n \frac{1}{i^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \dots + \frac{1}{n^2}$$

Dissenyeu l'algorisme en les tres estructures iteratives (condició inicial, final i repetitiva) i decidiu quina estructura és la més adient per aquest cas.

Programa en Pascal, estructura iterativa de condició final (REPEAT)

Falla en el cas $n = 0$ (sumatori de zero termes), ja que com a mínim la iteració s'executa un cop.

```

PROGRAM A6_2a;

VAR i, n : INTEGER;
    suma : REAL;

BEGIN
    WRITELN('Càlcul d''un sumatori');
    WRITELN('-----');

    (* Demanem el nombre del que volem calcular el sumatori. *)
    REPEAT
        WRITE('Introdueix el nombre de termes del sumatori : ');
        READLN(n);
        IF n < 0 THEN
            WRITELN('Nombre incorrecte. Torna-ho a provar.');
```

```
UNTIL n >= 0;

(* Calculem el sumatori *)
suma := 0;
i := 1;
REPEAT
  WRITELN('El terme ', i, ' és ', 1/(i*i):10:8);
  suma := suma + 1/(i*i);
  i := i + 1;
UNTIL i > n;

(* Donem el resultat *)
WRITELN('El resultat del sumatori és ', suma:10:8);
END.
```

Programa en Pascal, estructura iterativa de condició inicial (WHILE)

Ja no falla en el cas $n = 0$ (sumatori de zero termes), ja que la iteració pot no executar-se mai degut a que la condició està a l'inici del bucle.

```
PROGRAM A6_2a;

VAR i, n : INTEGER;
    suma : REAL;

BEGIN
  WRITELN('Càlcul d'un sumatori');
  WRITELN('-----');

  (* Demanem el nombre del que volem calcular el sumatori. *)
  REPEAT
    WRITE('Introdueix el nombre de termes del sumatori : ');
    READLN(n);
    IF n < 0 THEN
      WRITELN('Nombre incorrecte. Torna-ho a provar.');
```

Programa en Pascal, estructura iterativa repetitiva (FOR)

És la més adient, ja que coneixem el nombre de cops que s'haurà d'executar la iteració: n . Ens ho proporciona l'usuari pel teclat. Observeu la similitud de la seva estructura amb la de la fórmula del sumatori del problema.

```
PROGRAM A6_2c;

VAR i, n : INTEGER;
    suma : REAL;

BEGIN
```

```
WRITELN('Càlcul d'un sumatori');
WRITELN('-----');

(* Demanem el nombre del que volem calcular el sumatori. *)
REPEAT
  WRITE('Introdueix el nombre de termes del sumatori : ');
  READLN(n);
  IF n < 0 THEN
    WRITELN('Nombre incorrecte. Torna-ho a provar.');
```

```
UNTIL n >= 0;

(* Calculem el sumatori *)
suma := 0;
FOR i := 1 TO n DO
  BEGIN
    WRITELN('El terme ', i, ' és ', 1/(i*i):10:8);
    suma := suma + 1/(i*i);
  END;

(* Donem el resultat *)
WRITELN('El resultat del sumatori és ', suma:10:8);
END.
```

6.3 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que comprovi el valor d'una dada entera i després calculi el factorial de la dada.

- Aquesta dada ha d'ésser més gran que zero i més petita que vint.
- Si la dada no és correcta, l'ha de tornar a demanar tantes vegades com sigui necessari.

Recordem que el factorial d'un nombre enter és aquest nombre multiplicat per tots els seus antecessors:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$$

Vigileu amb els tipus de variables en Pascal, i el seu valor màxim de representació.

Programa en Pascal

```
PROGRAM A6_3;

VAR i , n : INTEGER;
    fact : LONGINT;

BEGIN
  WRITELN('Càlcul del factorial');
  WRITELN('-----');

  (* Demanem el nombre del que volem calcular el factorial, *)
  (* fins que sigui més gran que zero i més petit que 20. *)
  REPEAT
    WRITE('Introdueix un nombre entre 0 i 20 (no inclosos) : ');
    READLN(n);
    IF (n <= 0) OR (n >= 20) THEN
      WRITELN('Nombre incorrecte. Torna-ho a provar.');
```

```
UNTIL (n > 0) AND (n < 20);
```

```
(* Calculem el factorial *)
fact := 1;
FOR i := 2 TO n DO
    fact := fact * i;

(* També podíem haver calculat el factorial fent *)
(* compte enrere, amb les següents instruccions: *)
(* *)
(* fact := n; *)
(* FOR i := n-1 DOWNTO 2 DO *)
(*     fact := fact * i; *)

(* Donem el resultat *)
WRITELN('El factorial de ', n, ' és ', fact);
END.
```

6.4 Amplieu l'exercici 4.2 per tal que permeti calcular el centre de gravetat (la mitja) dels punts de l'espai bidimensional que l'ordinador demanarà per teclat fins que l'usuari introdueixi el punt origen de coordenades (0,0). Per aconseguir això, s'haurà d'acumular el valor de les coordenades i al mateix temps comptar el nombre de valors introduïts per l'usuari.

Programa en Pascal

```
PROGRAM A6_4;

VAR ax, ay, mx, my : REAL;
    terme: INTEGER;

BEGIN
    WRITELN('Càlcul del centre de gravetat d''una sèrie de punts');
    WRITELN('-----');

    (* Inicialitzem les variables *)
    mx := 0;
    my := 0;
    terme:=0;

    (* Llegim punts i els processem fins que *)
    (* introdueixin l'origen de coordenades. *)
    WRITE('Introdueix les coordenades d''un punt (0 0 per acabar): ');
    READLN(ax, ay);
    WHILE (ax <> 0) OR (ay <> 0) DO
        BEGIN
            mx := mx + ax;
            my := my + ay;
            terme := terme + 1;
            WRITE('Introdueix les coordenades d''un punt (0 0 per acabar): ');
            READLN(ax, ay);
        END;

    (* Hem de comprovar que hagin introduït algun terme *)
    (* abans de calcular la mitjana i mostrar els resultats *)
    IF terme > 0 THEN
        BEGIN
            mx := mx / terme;
            my := my / terme;
            WRITELN('Centre de gravetat: (', mx:4:2, ', ', my:4:2, ')');
        END;
    END;
```

```
END.
```

6.5 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal en els que l'usuari introdueixi nombres enters fins que endevini el nombre aleatori entre 0 i 100 generat inicialment per l'ordinador. El programa avisarà de que el nombre introduït per l'usuari és més gran o més petit que el nombre generat aleatòriament.

La instrucció de Pascal que ens permet generar un nombre aleatori és la següent:

`RANDOM(n)` genera un nombre enter aleatori entre 0 i $n-1$, ambdós inclosos.

La instrucció de Pascal que ens permet que els nombres aleatoris siguin diferents a cada nova execució del programa és la següent:

`RANDOMIZE` inicialitza el generador de nombres aleatoris.

Programa en Pascal

```
PROGRAM A6_5;

VAR x, n : INTEGER;

BEGIN
  RANDOMIZE;
  n := RANDOM(101);

  REPEAT
    WRITE('Introdueixi un nombre entre 0 i 100 : ');
    READLN(x);
    IF x < n THEN
      WRITELN('Més gran')
    ELSE
      IF x > n THEN
        WRITELN('Més petit')
      ELSE
        WRITELN('Has encertat !');
    UNTIL x = n;
  END.
```

6.6 A la botiga dels germans Pastafullada és tradició presentar les llaunes de conserva apilades triangularment. En el primer pis n'hi ha una, en el segon dues, en el tercer tres i així successivament. Per exemple, sis llaunes es posen així:

```
      *
     * *
    * * *
```

Els germans tenen grans problemes per fer les comandes de llaunes. Fixeu-vos que no tot nombre de llaunes pot apilar-se triangularment: per exemple, 8 llaunes no poden apilar-se. Escriviu un algorisme en pseudocodi i el corresponent

programa en Pascal tal que, donat un nombre natural, comprova si aquest és adequat per muntar piles.

Programa en Pascal

```
PROGRAM A6_6;

VAR n , i : INTEGER;

BEGIN
  REPEAT
    WRITE('Introdueixi un nombre de llaunes més gran que zero : ');
    READLN(n);
  UNTIL n > 0;

  (* Comencem a apilar. Restarem quantitats successives de llaunes *)
  (* a les llaunes que tenim: primer una, després dues, i així *)
  (* successivament fins que no quedi cap. *)
  i := 1;
  WHILE n > 0 DO
    BEGIN
      n := n - i;
      i := i + 1;
    END;

  (* Si al final ens han quedat zero llaunes justes era apilable. *)
  IF n = 0 THEN
    WRITELN('La quantitat és apilable triangularment')
  ELSE
    WRITELN('La quantitat no és apilable triangularment');
END.
```

6.7 Dissenyau un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que calculi i visualitzi els elements de la sèrie de Fibonacci. Aquesta sèrie es defineix pels seus paràmetres:

$$\begin{aligned} \text{Fibonacci}(0) &= 1 \\ \text{Fibonacci}(1) &= 1 \\ \text{Fibonacci}(n) &= \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2) \end{aligned}$$

L'usuari haurà d'entrar, només, el nombre d'elements que vol visualitzar.

Programa en Pascal

```
PROGRAM A6_7;

VAR n , i : INTEGER;
    ant2 : LONGINT;      (* Fibonacci(n-2) *)
    ant1 : LONGINT;      (* Fibonacci(n-1) *)
    fibo : LONGINT;      (* Fibonacci(n) *)

BEGIN
  REPEAT
    WRITE('Introdueixi quants elements vol visualitzar : ');
    READLN(n);
  UNTIL n > 0;

  (* Inicialitzem les variables *)
```

```

ant2 := 1;          (* Fibonacci(0) = 1 *)
ant1 := 1;          (* Fibonacci(1) = 1 *)

(* Calculem els termes de la sèrie de fibonacci *)
FOR i := 0 TO n-1 DO
  BEGIN
    IF (i = 0) OR (i = 1) THEN
      fibo := 1
    ELSE
      BEGIN
        fibo := ant1 + ant2;
        ant2 := ant1;
        ant1 := fibo;
      END;
    Writeln('Fibonacci(', i, ') = ', fibo);
  END;
END.

```

6.8 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que, introduïts dos nombres enters a i b per teclat, trobi el resultat de realitzar la seva multiplicació a partir de sumes. És a dir:

$$a \times b = a + a + a + \dots + a \quad (a \text{ sumat } b \text{ vegades})$$

Tingueu en compte que tant a com b poden ser nombres negatius.

Programa en Pascal

```

PROGRAM A6_8;

VAR a, b, i : INTEGER;
    mult     : LONGINT;

BEGIN
  Writeln('Càlcul del producte d'enters a partir de sumes');
  Writeln('-----');

  Write('Introdueix el primer nombre : ');
  Readln(a);
  Write('Introdueix el segon nombre : ');
  Readln(b);

  (* Podem optimitzar el codi si ara afegim les següents línies, *)
  (* que intercanviaran els valors de a i b si a < b. D'aquesta *)
  (* manera sumarem a cops b, en lloc de b cops a. Cal haver *)
  (* declarat prèviament una variable 'aux' de tipus 'integer'. *)
  (* *)
  (* IF a < b THEN *)
  (*   BEGIN *)
  (*   aux := a; *)
  (*   a := b; *)
  (*   b := aux; *)
  (*   END; *)

  (* Fem el càlcul del producte a partir de sumes, *)
  (* tenint en compte que 'b' pot ser negatiu. *)
  mult := 0;
  IF b >= 0 THEN
    FOR i := 1 TO b DO
      mult := mult + a
    ELSE

```

```

    FOR i := 1 TO -b DO
        mult := mult - a;

    (* Una altra manera d'haver tingut en compte que b podia ser *)
    (* negatiu era calcular la multiplicació de la següent manera *)
    (* *)
    (* mult := 0; *)
    (* FOR i := 1 TO ABS(b) DO *)
    (*     mult := mult + a; *)
    (* IF b < 0 THEN *)
    (*     mult := -mult; *)

    (* Una altra manera d'haver tingut en compte que b podia ser *)
    (* negatiu era calcular la multiplicació de la següent manera *)
    (* *)
    (* mult := 0; *)
    (* IF b < 0 THEN *)
    (*     BEGIN *)
    (*         a := -a; *)
    (*         b := -b; *)
    (*     END; *)
    (* FOR i := 1 TO b DO *)
    (*     mult := mult + a; *)

    (* Donem el resultat *)
    WRITELN('El producte de ', a, ' per ', b, ' és ', mult);
END.
```

6.9 A un ordinador primitiu construït a inici dels 80 cal fer el càlcul de la funció $f(x) = \sin(x)$ per alguns valors de x . A tal efecte cal utilitzar el desenvolupament en sèrie de Taylor

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots$$

aturant el càlcul serà quan el terme calculat, en valor absolut, sigui més petit o igual que un valor d'error ε donat per l'usuari o fixat pel programa ($\varepsilon \geq |x^n / n!|$). Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que efectuï el càlcul. Tingueu en compte que les unitats de l'angle les haurem de passar a radians per poder realitzar les operacions.

Programa en Pascal

```

PROGRAM A6_9;

VAR x      : REAL;
    error  : REAL;
    sinus  : REAL;
    terme  : REAL;
    i      : INTEGER;

BEGIN
    REPEAT
        WRITE('Introdueixi un angle en graus i calcularé el seu sinus : ');
        READLN(x);
    UNTIL (x >= -360) AND (x <= 360);

    (* Passem els graus a radians *)
```

```
x := x*PI/180;

REPEAT
  WRITE('Introdueixi l'error màxim permès : ');
  READLN(error);
UNTIL error > 0;

(* Inicialitzem les variables *)
i := 1;
terme := x;
sinus := x;

(* Calculem el sinus fins que l'error sigui prou petit *)
WHILE ABS(terme) > error DO
  BEGIN
    i := i + 2;
    terme := - terme * x*x/(i*(i-1));
    sinus := sinus + terme;
  END;

  WRITELN('El sinus de ', x, ' radians és ', sinus);
END.
```

6.10 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que calculi l'arrel quadrada d'un nombre real positiu a qualsevol mitjançant el desenvolupament en sèrie

$$x_1 = a$$

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{a}{x_n} \right)$$

i que aturi el càlcul quan dues aproximacions successives difereixin en menys d'un valor ϵ donat per l'usuari o fixat pel programa.

Programa en Pascal

```
PROGRAM A6_10;

VAR a, error, terme_ant, terme_seg : REAL;

BEGIN
  REPEAT
    WRITE('De quin nombre vol calcular l'arrel quadrada ? ');
    READLN(a);
    IF a <= 0 THEN
      WRITELN('Ha de ser un nombre positiu més gran que zero.');
```

```
    terme_seg := (terme_ant + a/terme_ant)/2;  
    UNTIL ABS(terme_seg - terme_ant) < error;  
  
    Writeln('L'arrel quadrada de ' , a , ' és ' , terme_seg);  
END.
```

Pràctica 7. Variables estructurades homogènies

7.1 Sense ajuda de l'ordinador, determineu el valor dels vectors a i b a cada pas de l'execució de les següents instruccions:

VAR a, b: ARRAY [1..3] OF INTEGER;	->	1 2 3	a ? ? ?	1 2 3	b ? ? ?	
i: INTEGER;	->		a ? ? ?		b ? ? ?	i ?
BEGIN						
a[1] := 2;	->		a 2 ? ?		b ? ? ?	i ?
a[2] := 4;	->		a 2 4 ?		b ? ? ?	i ?
a[3] := a[1] + a[2];	->		a 2 4 6		b ? ? ?	i ?
b[2 + 1] := a[1] - 1;	->		a 2 4 6		b ? ? 1	i ?
b[a[1]] := 2*a[2] - 1;	->		a 2 4 6		b ? 7 1	i ?
b[a[3]/(a[1] + a[2])] := b[1] + 1;	->		incorrecte!		index no enter!	
FOR i := 1 TO 3 DO b[i] := b[i] + i;	->		a 2 4 6		b ? 9 4	i 3
b[i+1] := 9;	->		a 2 4 6		b ? 9 4	i 9
END.						

Descobriu l'error semàntic i els dos errors de concepte que amaga el programa.

Error semàntic (instrucció 6): l'índex no pot ser un real.

Error de concepte (instruccions 6 i 7): referen. b[1] sense estar inicialitzat.

Error de concepte (instrucció 6): desbordament, b[4] se solapa amb i.

7.2 Observeu el següent programa en Pascal, que primer llegeix un vector de deu elements numèrics i a continuació calcula el seu mòdul, segons la fórmula:

$$\text{sigui } \vec{v} = (\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n) \text{ llavors } |\vec{v}| = \sqrt{\vec{v}_1^2 + \vec{v}_2^2 + \dots + \vec{v}_n^2}$$

Responeu les següents qüestions:

- Quina diferència hi ha entre una constant (CONST), un tipus de dades (TYPE) i una variable (VAR)?

Una declaració de constant associa una etiqueta a un valor, i durant la compilació tota referència a l'etiqueta queda substituïda pel seu valor corresponent (podem veure una constant com una variable que no es pot modificar i que internament no ocupa espai a memòria).

Una declaració de tipus associa una etiqueta a la descripció d'un nou tipus de dades, i durant la compilació tota referència a l'etiqueta a qualsevol declaració de variables queda substituïda per la descripció del tipus de dades corresponent.

Una declaració de variable associa una etiqueta a unes posicions de memòria, i durant la compilació tota referència a l'etiqueta queda substituïda per la posició de memòria corresponent o pel seu contingut.

- Quina és l'avantatge de declarar constants? **El codi del programa queda més llegible (desapareixen els nombres "màgics") i és més fàcil realitzar modificacions (amb canviar el valor de la constant bastà, no cal cercar el valor per tot el codi del programa).**
- Quina és l'avantatge de declarar nous tipus de dades? **La declaració de variables queda més breu i llegible, i podem crear nous tipus de dades a partir d'altres ja creats.**

Compileu el programa i realitzeu una execució pas per pas (tecla F7) amb traça de variables (tecles CTRL+F7), per tal de visualitzar en tot moment l'estat dels vectors.

Programa en Pascal

```
PROGRAM A7_2;

CONST
  MAX_ELEMENTS = 10;

TYPE
  t_vector = ARRAY [1..MAX_ELEMENTS] OF REAL;

VAR
  v : t_vector;
  i : INTEGER;
  suma : REAL;

BEGIN
  WRITELN('Càlcul del mòdul d'un vector');
  WRITELN('-----');
  WRITELN;

  (* Omplim el vector *)
  FOR i := 1 TO MAX_ELEMENTS DO
    BEGIN
      WRITE('Introdueix l'element ', i, ' : ');
      READLN(v[i]);
    END;

  (* Calculem la suma dels quadrats *)
  suma := 0;
  FOR i := 1 TO MAX_ELEMENTS DO
    BEGIN
      suma := suma + v[i]*v[i];
    END;

  (* Imprimim el mòdul *)
  WRITELN('El mòdul del vector és ', SQRT(suma));
END.
```

7.3 Dissenyau un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que emmagatzemi en un vector la nota dels alumnes d'un grup de pràctiques i posteriorment calculi i visualitzi el nombre de notes que apareixen dins dels següents intervals:

[0 , 5[Insuficient
[5 , 7[Aprovat
[7 , 9[Notable
[9 , 10]	Excel·lent

Tingueu en compte que, encara que els grups de pràctiques tenen un màxim de vint alumnes, cada grup pot tenir un nombre d'alumnes diferent. El programa ha de servir per qualsevol grup.

Programa en Pascal

```
PROGRAM A7_3;

CONST MAX_ALUMNES = 20;

VAR notes : ARRAY [1..MAX_ALUMNES] OF REAL;
    alumne, num_alumnes : 1..MAX_ALUMNES;
    insuficients, aprovats, notables, excellents : INTEGER;

BEGIN
    (* Llegim quants alumnes té el grup *)
    REPEAT
        WRITE('Quants alumnes té el grup (de 1 a ', MAX_ALUMNES, ')? ');
        READLN(num_alumnes);
    UNTIL (num_alumnes >= 1) AND (num_alumnes <= MAX_ALUMNES);

    (* Llegim les notes de tots els alumnes *)
    FOR alumne := 1 TO num_alumnes DO
        REPEAT
            WRITE('Introdueix la nota de l''alumne ', alumne, ' : ');
            READLN(notes[alumne]);
        UNTIL (notes[alumne] >= 0) AND (notes[alumne] <= 10);

        (* Contem el nombre de suspensos, aprovats, notables i excel·lents *)
        insuficients := 0;
        aprovats := 0;
        notables := 0;
        excellents := 0;

        FOR alumne := 1 TO num_alumnes DO
            BEGIN
                IF notes[alumne] < 5 THEN
                    insuficients := insuficients + 1
                ELSE
                    IF notes[alumne] < 7 THEN
                        aprovats := aprovats + 1
                    ELSE
                        IF notes[alumne] < 9 THEN
                            notables := notables + 1
                        ELSE
                            excellents := excellents + 1;
                END;
            END;

    (* Donem els resultats *)
    Writeln('Insuficients: ', Insuficients, '    Aprovats: ', aprovats,
        '    Notables: ', notables, '    Excel·lents: ', excellents);
```

END.

7.4 Amplieu l'exercici 6.4 per tal que permeti trobar el centre de gravetat d'un conjunt de punts de l'espai bidimensional emmagatzemats a un vector, segons la fórmula:

Sigui el vector de punts $\vec{v} = ((v_{1x}, v_{1y}), \dots, (v_{nx}, v_{ny}))$ llavors $\vec{m} = \sum_{i=1}^n \vec{v}_i = \left(\frac{\vec{v}_{1x} + \dots + \vec{v}_{nx}}{n}, \frac{\vec{v}_{1y} + \dots + \vec{v}_{ny}}{n} \right)$

L'usuari indicarà a inici de programa quants punts vol introduir.

Programa en Pascal

```
PROGRAM A7_4;

CONST MaxElements = 10;

VAR punt_x, punt_y : ARRAY [1 .. MaxElements] OF REAL;
    NumElements, posicio : 1..MaxElements;
    mx, my : REAL;

BEGIN
    WRITELN('Centre de gravetat dels punts guardats en un vector');
    WRITELN('-----');

    (* Llegim quants punts vol introduir *)
    REPEAT
        WRITE('Quants punts vols introduir (de 1 a ', MaxElements, '? ');
        READLN(NumElements);
    UNTIL (NumElements >= 1) AND (NumElements <= MaxElements);

    (* Llegim les coordenades x i y dels punts *)
    FOR posicio := 1 TO NumElements DO
        BEGIN
            WRITE('Introdueix les coordenades del punt ', posicio, ' : ');
            READLN(punt_x[posicio], punt_y[posicio]);
        END;

    (* Calculem el punt mig *)
    mx := 0;
    my := 0;
    FOR posicio := 1 TO NumElements DO
        BEGIN
            mx := mx + punt_x[posicio];
            my := my + punt_y[posicio];
        END;
    mx := mx / NumElements;
    my := my / NumElements;

    (* Mostrem el punt mig *)
    WRITELN('Centre de gravetat: ( ', mx:4:2, ', ', my:4:2, ' )');
END.
```

7.5 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que sigui capaç de llegir els valors d'una matriu 3×3 i calculi el valor del determinant.

Programa en Pascal

```
PROGRAM A7_5;

VAR
  i , j : INTEGER;
  m      : ARRAY [ 1..3 , 1..3 ] OF REAL;
  det    : REAL;

BEGIN
  WRITELN('Determinant d''una matriu de dimensions 3 x 3');
  WRITELN('-----');

  (* Omplim la matriu *)
  WRITELN;
  FOR i := 1 TO 3 DO
    FOR j := 1 TO 3 DO
      BEGIN
        WRITE('Introdueixi l''element [ ' , i , ' , ' , j ,
              ' ] de la matriu : ');
        READLN(m[i,j]);
      END;
    END;

  (* Calculem el determinant *)
  det := m[1,1]*m[2,2]*m[3,3] + m[1,2]*m[2,3]*m[3,1]
        + m[1,3]*m[2,1]*m[3,2] - m[1,1]*m[2,3]*m[3,2]
        - m[1,2]*m[2,1]*m[3,3] - m[1,3]*m[2,2]*m[3,1];

  (* Imprimim el resultat *)
  WRITELN;
  WRITELN('El determinant és ', det:5:3);
END.
```

7.6 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que ens permeti realitzar la suma de dues matrius de dimensions $N \times N$ (utilitzarem un màxim de 10×10). La suma ens ve donada per la següent fórmula:

$$R[i, j] = M_1[i, j] + M_2[i, j]$$

Un cop escrit i compilat el programa, realitzeu una execució pas per pas amb traça de les variables, per veure el funcionament dels bucles FOR niats.

Què ens cal canviar a l'anterior algorisme o programa per tal que en lloc de sumar matrius les multipliqui? La multiplicació ens ve donada per la següent fórmula:

$$R[i, j] = \sum_{k=1}^n M_1[i, k] \times M_2[k, j]$$

Programa en Pascal

```
PROGRAM A7_6;
```

```

CONST
  MAX_D = 10;          (* Màxim de files i columnes de la matriu *)

VAR
  i , j, d      : 1..MAX_D;
  R , M1 , M2  : ARRAY [ 1..MAX_D , 1..MAX_D ] OF REAL;

BEGIN
  WRITELN('Suma de dues matrius de dimensions N x N');
  WRITELN('-----');

  (* Demanem el nombre de files i columnes de les matrius *)
  WRITELN;
  REPEAT
    WRITE('Introdueixi la dimensió de les matrius : ');
    READLN(d);
  UNTIL (d >= 1) AND (d <= MAX_D);

  (* Omplim la primera matriu *)
  WRITELN;
  FOR i := 1 TO d DO
    FOR j := 1 TO d DO
      BEGIN
        WRITE('Introdueixi l''element [', i , ', ' , j ,
              '] de la primera matriu : ');
        READLN(M1[i,j]);
      END;
    END;

  (* Omplim la segona matriu *)
  WRITELN;
  FOR i := 1 TO d DO
    FOR j := 1 TO d DO
      BEGIN
        WRITE('Introdueixi l''element [', i , ', ' , j ,
              '] de la segona matriu : ');
        READLN(M2[i,j]);
      END;
    END;

  (* Calculem la matriu resultat *)
  FOR i := 1 TO d DO
    FOR j := 1 TO d DO
      BEGIN
        R[i,j] := M1[i,j] + M2[i,j];
      END;
    END;

  (* Imprimim la matriu resultat *)
  WRITELN;
  WRITELN('La matriu suma és:');
  FOR i := 1 TO d DO
    BEGIN
      WRITE('[');
      FOR j := 1 TO d DO WRITE(R[i,j]:9:2);
      WRITELN(' ]');
    END;
  END;
END.

```

Les instruccions que cal canviar al programa anterior per tal que, en lloc de sumar matrius les multipliqui, són les següents:

- El codi que calculava l'element ij de la matriu resultat era:

```
R[i,j] := M1[i,j] * M2[i,j];
```

- El nou codi per calcular aquest element, que substituirà el codi anterior, serà (caldrà definir la variable k):

```
R[i,j] := 0;  
FOR k := 1 TO d DO  
  R[i,j] := R[i,j] + M1[i,k]*M2[k,j];
```

7.7 Dissenyeu un algorisme en pseudocodi i el corresponent programa en Pascal que donat un vector de 50 elements enters, el descompongui en dos, un format pels valors senars i un altre pels valors parells. Cal remarcar que en aquests dos vectors resultants els valors es posaran correlativament (un després de l'altre).

Programa en Pascal

```
PROGRAM A7_7;  
  
CONST NUM_EL = 50;  
  
VAR vector   : ARRAY [1..NUM_EL] OF INTEGER;   (* Vector d'elements *)  
    senars   : ARRAY [1..NUM_EL] OF INTEGER;   (* Vector de senars *)  
    parells  : ARRAY [1..NUM_EL] OF INTEGER;   (* Vector de parells *)  
    num_senars : INTEGER;                       (* Nombre de senars *)  
    num_parells : INTEGER;                      (* Nombre de parells *)  
    i         : INTEGER;                        (* Comptador bucles *)  
  
BEGIN  
  (* Llegim els elements *)  
  FOR i := 1 TO NUM_EL DO  
    BEGIN  
      WRITE('Introdueixi l''element ', i, ' del vector : ');  
      READLN(vector[i]);  
    END;  
  
  (* Inicialitzem les variables *)  
  num_senars := 0;  
  num_parells := 0;  
  
  (* Separem els elements en senars i parells *)  
  FOR i := 1 TO NUM_EL DO  
    IF (VECTOR[i] MOD 2) = 0 THEN  
      BEGIN  
        num_parells := num_parells + 1;  
        parells[num_parells] := vector[i];  
      END  
    ELSE  
      BEGIN  
        num_senars := num_senars + 1;  
        senars[num_senars] := vector[i];  
      END;  
  
  (* Imprimim els vectors de senars i de parells *)  
  WRITELN;  
  WRITELN('Els senars són : ');  
  FOR i := 1 TO num_senars DO  
    WRITELN(senars[i]);  
  WRITELN;  
  WRITELN('Els parells són : ');  
  FOR i := 1 TO num_parells DO  
    WRITELN(parells[i]);  
END.
```

7.8 Dissenyeu un algorisme en pseudocodi i el corresponent programa en Pascal que donat un vector de 15 elements amb valors aleatoris introduïts per l'usuari, sigui capaç d'ordenar el vector i treure el resultat per pantalla.

Programa en Pascal

```
PROGRAM A7_8;

CONST NUM_EL = 15;

TYPE t_element = INTEGER;

VAR vector : ARRAY [1..NUM_EL] OF t_element; (* Vector d'elements
*)
    aux : t_element; (* Variable per intercanviar
*)
    i, j : INTEGER; (* Comptadors per bucles
*)
    pos_petit : INTEGER; (* Posició element més petit
*)

BEGIN
    (* Llegim els elements *)
    FOR i := 1 TO NUM_EL DO
        BEGIN
            WRITE('Introdueixi l'element ', i, ' del vector : ');
            READLN(vector[i]);
            END;

    (* Hi han molts mètodes per ordenar un vector. Farem servir el *)
    (* d'ordenació directa perquè és el més intuïtiu. Consisteix en: *)
    (* per cada element del vector, des del primer fins el penúltim, *)
    (* cercar en els restants elements que queden per ordenar la *)
    (* posició de l'element més petit. Un cop trobada, intercanviar-los. *)
    (* Per l'últim element del vector no cal fer aquesta operació, ja *)
    (* que si tots els elements anteriors estan ordenats, l'últim també *)
    (* ho estarà. *)

    (* Per a cada element del vector excepte l'últim *)
    FOR i := 1 TO NUM_EL-1 DO
        BEGIN
            (* Trobar la posició de l'element més petit *)
            (* dels elements que resten per ordenar *)
            pos_petit := i;
            FOR j := i+1 TO NUM_EL DO
                IF vector[j] < vector[pos_petit] THEN
                    pos_petit := j;
            (* Intercanviar dins el vector l'element a *)
            (* ordenar amb l'element més petit trobat *)
            aux := vector[i];
            vector[i] := vector[pos_petit];
            vector[pos_petit] := aux;
            END;

    (* Imprimim el vector ordenat *)
    WRITELN;
    WRITELN('Els vector ordenat és : ');
    FOR i := 1 TO NUM_EL DO
        WRITELN(vector[i]);
    END.
```

7.9 Dissenyeu un algorisme en pseudocodi i el corresponent programa en Pascal que donats dos vectors ordenats realitzi la fusió d'ambdós per obtenir un tercer vector també ordenat. Cada vector conté cinc elements.

Programa en Pascal

```
PROGRAM A7_9;

CONST
  MAX_ELEMENTS = 5;          (* Nombre d'elements dels vectors *)

VAR
  posicio_1 , posicio_2 , posicio_r : INTEGER;
  vector1 , vector2 : ARRAY [ 1 .. MAX_ELEMENTS ] OF INTEGER;
  vectorR          : ARRAY [ 1 .. 2*MAX_ELEMENTS ] OF INTEGER;

BEGIN
  WRITELN('Fusió de dos vectors ordenats');
  WRITELN('-----');
  WRITELN;

  WRITELN('Ara haurà d'introduir els dos vectors ordenats.');
```

WRITELN('Cada vector conté ', MAX_ELEMENTS, ' elements.');

(* Omplim el primer vector *)

```
WRITELN;
FOR posicio_1 := 1 TO MAX_ELEMENTS DO
  BEGIN
    WRITE('Introdueix el valor de la posició ', posicio_1, ' : ');
    READLN(vector1[posicio_1]);
  END;
```

(* Omplim el segon vector *)

```
WRITELN;
FOR posicio_2 := 1 TO MAX_ELEMENTS DO
  BEGIN
    WRITE('Introdueix el valor de la posició ', posicio_2, ' : ');
    READLN(vector2[posicio_2]);
  END;
```

(* Realitzem la fusió dels dos vectors *)

```
posicio_1 := 1;
posicio_2 := 1;
posicio_r := 1;
REPEAT
  IF vector1[posicio_1] <= vector2[posicio_2] THEN
    BEGIN
      vectorR[posicio_r] := vector1[posicio_1];
      posicio_r := posicio_r + 1;
      posicio_1 := posicio_1 + 1;
      IF posicio_1 > MAX_ELEMENTS THEN
        REPEAT
          vectorR[posicio_r] := vector2[posicio_2];
          posicio_2 := posicio_2 + 1;
          posicio_r := posicio_r + 1;
        UNTIL posicio_2 > MAX_ELEMENTS;
    END
  ELSE
    BEGIN
      vectorR[posicio_r] := vector2[posicio_2];
```

```

    posicio_r := posicio_r + 1;
    posicio_2 := posicio_2 + 1;
    IF posicio_2 > MAX_ELEMENTS THEN
        REPEAT
            vectorR[posicio_r] := vector1[posicio_1];
            posicio_1 := posicio_1 + 1;
            posicio_r := posicio_r + 1;
        UNTIL posicio_1 > MAX_ELEMENTS;
    END;
UNTIL posicio_r > MAX_ELEMENTS*2;

(* Imprimim el vector fusionat *)
WRITELN;
WRITELN('El vector fusionat és :');
FOR posicio_r := 1 TO 2*MAX_ELEMENTS DO
    WRITELN('L'element de la posició ', posicio_r:2, ' és ',
            vectorR[posicio_r]:6);
END.

```

7.10 Suposem que disposem d'una taula amb distàncies quilomètriques:

	Barcelona	Girona	Lleida	Tarragona	Saragossa	Terol
Barcelona		100	156	98	296	409
Girona			256	198	396	509
Lleida				91	140	319
Tarragona					231	311
Saragossa						181
Terol						

Dissenyeu un algorisme en pseudocodi i el corresponent programa en Pascal que ens permeti calcular:

- ┆ La distància entre dues poblacions, el nom de les quals serà introduït per l'usuari.
- ┆ Les dues ciutats més allunyades entre si i la distància que les separa.
- ┆ La distància total recorreguda en l'itinerari circular que passa per totes les ciutats en l'ordre següent: primera, segona, tercera, ..., última i primera de nou.

Programa en Pascal

```

PROGRAM A7_10;

CONST
    (* Nombre de ciutats *)
    NUM_CIUATATS = 6;

    (* Noms de les ciutats *)
    CIUTATS : ARRAY [1..NUM_CIUATATS] OF STRING =
        ('BARCELONA', 'GIRONA', 'LLEIDA', 'TARRAGONA', 'SARAGOSSA',
        'TEROL');

    (* Distància entre ciutats *)
    DIST : ARRAY [1..NUM_CIUATATS,1..NUM_CIUATATS] OF INTEGER =

```

```
( ( 0, 100, 156, 98, 296, 409),
  (100, 0, 256, 198, 396, 509),
  (156, 256, 0, 91, 140, 319),
  ( 98, 198, 91, 0, 231, 311),
  (296, 396, 140, 231, 0, 181),
  (409, 509, 319, 311, 181, 0));

VAR
  ciutat      : STRING;
  i, j        : INTEGER;
  iMax, jMax  : INTEGER;
  distancia   : INTEGER;

BEGIN
  (* Llegim el nom de la primera ciutat *)
  REPEAT
    WRITE('Introdueixi el nom de la primera ciutat : ');
    READLN(ciutat);
    (* Passem el nom a majúscules *)
    FOR i := 1 to LENGTH(ciutat) DO
      ciutat[i] := UPCASE(ciutat[i]);
    (* Busquem la posició de la ciutat *)
    i := 1;
    WHILE (i <= NUM_CIUTATS) AND (ciutat <> CIUTATS[i]) DO
      i := i + 1;
  UNTIL (i >= 1) AND (i <= NUM_CIUTATS);

  (* Llegim el nom de la segona ciutat *)
  REPEAT
    WRITE('Introdueixi el nom de la segona ciutat : ');
    READLN(ciutat);
    (* Passem el nom a majúscules *)
    FOR j := 1 to LENGTH(ciutat) DO
      ciutat[j] := UPCASE(ciutat[j]);
    (* Busquem la posició de la ciutat *)
    j := 1;
    WHILE (j <= NUM_CIUTATS) AND (ciutat <> CIUTATS[j]) DO
      j := j + 1;
  UNTIL (j >= 1) AND (j <= NUM_CIUTATS);

  (* Imprimim la distància *)
  WRITELN('La distància és ', DIST[i,j],
    ' entre ', CIUTATS[i], ' i ', CIUTATS[j]);

  (* Imprimim les ciutats més distants *)
  iMax := 1;
  jMax := 1;
  FOR i := 1 TO NUM_CIUTATS-1 DO
    FOR j := i+1 TO NUM_CIUTATS DO
      IF DIST[i,j] > DIST[iMax,jMax] THEN
        BEGIN
          iMax := i;
          jMax := j;
        END;
  WRITELN('La distància més gran entre ciutats és ', DIST[iMax,jMax],
    ' entre ', CIUTATS[iMax], ' i ', CIUTATS[jMax]);

  (* Imprimim la distància d'un itinerari circular *)
  distancia := 0;
  WRITE('La distància d'un itinerari circular (', CIUTATS[1], ' -> ');
  FOR j := 2 TO NUM_CIUTATS DO
    BEGIN
      distancia := distancia + DIST[j-1,j];
      WRITE(CIUTATS[j], ' -> ');
    END;
  distancia := distancia + DIST[1,j];
  WRITELN(CIUTATS[1], ') és ', distancia);
```

END.

7.11 Un/a alumne d'enginyeria desitja realitzar una estadística de les hores d'estudi mensuals dedicades a cadascuna de les seves assignatures. Dissenyeu un algorisme en pseudocodi i el corresponent programa en Pascal que ens permeti calcular:

- 1 El total anual d'hores dedicades a cada assignatura.
- 1 El total mensual d'hores dedicades a estudiar.
- 1 El nom i total d'hores de l'assignatura més estudiada.

	Gener	Febrer	...	Decembre	Total
Assignatura 1					
...					
Assignatura 5					
Total					

Programa en Pascal

```
PROGRAM A7_11;

CONST
  MAX_ASS = 5;           (* Màxim nombre de files *)
  MAX_MES = 12;          (* Màxim nombre de columnes *)

VAR
  num_ass , ass : 1..MAX_ASS;
  num_mes , mes : 1..MAX_MES;
  matriu        : ARRAY [ 1..MAX_ASS , 1..MAX_MES ] OF REAL;
  suma_ass      : ARRAY [ 1..MAX_ASS ] OF REAL;
  suma_mes      : ARRAY [ 1..MAX_MES ] OF REAL;
  maxim         : REAL;
  maxim_ass     : 1..MAX_ASS;

BEGIN
  WRITELN('Estadística anual d''hores d''estudi');
  WRITELN('-----');
  WRITELN;

  (* Demanem el nombre d'assignatures *)
  REPEAT
    WRITE('Introdueixi el nombre d'assignatures (1..' ,MAX_ASS,' ) : ');
    READLN(num_ass);
  UNTIL (num_ass >= 1) AND (num_ass <= MAX_ASS);

  (* El nombre de mesos el suposem fixe *)
  num_mes := MAX_MES;

  (* Omplim la matriu *)
  WRITELN;
  FOR ass := 1 TO num_ass DO
    FOR mes := 1 TO num_mes DO
      BEGIN
        WRITE('Introdueixi les hores d''estudi de l''assignatura ', ass,
              ' durant el mes ', mes, ' : ');
        READLN(matriu[ass,mes]);
      END
    END
  END

```

```

        END;

        (* Calculem la suma per files i per columnes *)
        FOR ass := 1 TO num_ass DO suma_ass[ass] := 0;
        FOR mes := 1 TO num_mes DO suma_mes[mes] := 0;

        FOR ass := 1 TO num_ass DO
            FOR mes := 1 TO num_mes DO
                BEGIN
                    suma_ass[ass] := suma_ass[ass] + matriu[ass,mes];
                    suma_mes[mes] := suma_mes[mes] + matriu[ass,mes];
                END;
            END;

        (* Imprimim la matriu i la suma de les files i les columnes. *)
        WRITELN;

        FOR ass := 1 TO num_ass DO
            BEGIN
                FOR mes := 1 TO num_mes DO WRITE(matriu[ass,mes]:6:1);
                WRITELN(' |' , suma_ass[ass]:5:1);
            END;

        FOR mes := 1 TO num_mes DO WRITE('-----');
        WRITELN('-+');

        FOR mes := 1 to num_mes DO WRITE(suma_mes[mes]:6:1);
        WRITELN;

        (* Calculem l'assignatura amb total màxim *)
        maxim_ass := 1;

        FOR ass := 2 TO num_ass DO
            IF suma_ass[ass] > suma_ass[maxim_ass] THEN
                maxim_ass := ass;
            END;

        (* Imprimim l'assignatura amb total màxim *)
        WRITELN;
        WRITELN('L'assignatura ', maxim_ass, 'té el màxim d''hores d'
            'estudi anuals : ', suma_ass[maxim_ass]:5:1);
    END.

```

7.12 Una matriu quasi-nul·la és una matriu amb un alt percentatge d'elements nuls. Una matriu quasi-nul·la amb k elements no nuls se sol representar emmagatzemant els elements no nuls en una matriu de k files i tres columnes, contenint cada columna d'aquesta matriu la fila, la columna i el valor dels elements no nuls, respectivament. Per exemple:

$$\text{la matriu quasi-nul·la } \begin{pmatrix} 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ es pot representar per } \begin{pmatrix} 1 & 3 & 3 \\ 3 & 2 & 6 \\ 3 & 3 & 1 \\ 5 & 5 & 1 \end{pmatrix}$$

Dissenyeu un algorisme en pseudocodi i el corresponent programa en Pascal per tal de convertir una matriu quasi-nul·la en representació normal a la nova representació més compacta.

Programa en Pascal

```
PROGRAM A7_12;

CONST
  MAX_FIL = 10;          (* Màxim nombre de files de la matriu *)
  MAX_COL = 10;         (* Màxim nombre de columnes de la matriu *)
  MAX_EL = (MAX_FIL*MAX_COL*20) DIV 100; (* # Elem matriu quasi-nul.la *)

VAR
  i, j, k : INTEGER;
  fil, col : INTEGER;
  M : ARRAY [1..MAX_FIL, 1..MAX_COL] OF REAL;
  el : INTEGER;
  Mc : ARRAY [1..MAX_EL, 1..3] OF REAL;

BEGIN
  WRITELN('Representació compacta de matrius quasi-nul.lles');
  WRITELN('-----');

  (* Demanem el nombre de files i columnes de la matriu *)
  WRITELN;
  REPEAT
    WRITE('Introdueixi les files i columnes de la matriu: ');
    READLN(fil, col);
  UNTIL (fil >= 1) AND (fil <= MAX_FIL) AND (col >= 1) AND (col <=
MAX_COL);

  (* Omplim la matriu *)
  WRITELN;
  FOR i := 1 TO fil DO
    FOR j := 1 TO col DO
      BEGIN
        WRITE('Introdueixi l''element [' , i , ',' , j , '] de la matriu: ');
      );
      READLN(M[i,j]);
      END;

  (* Passem a representació compacta la matriu *)
  el := 0;
  FOR i := 1 TO fil DO
    FOR j := 1 TO col DO
      IF M[i,j] <> 0 THEN
        BEGIN
          el := el + 1;
          Mc[el,1] := i;
          Mc[el,2] := j;
          Mc[el,3] := M[i,j];
        END;

  (* Imprimim la matriu en representació compacta *)
  WRITELN;
  WRITELN('La matriu compacta és:');
  FOR k := 1 TO el DO
    WRITELN([' , Mc[k,1]:2:0, ' , ' , Mc[k,2]:2:0, ' , ' , Mc[k,3]:6:2,
']');
  END.
```

7.13 Dissenyau un algorisme en pseudocodi i el corresponent programa en Pascal que donat un nombre introduït en una base qualsevol b_1 sigui capaç de

convertir-lo a una altra base qualsevol b_2 . Com que ens veiem limitats a l'hora de treballar amb bases pel nombre de símbols de que disposem, emprarem els símbols 0, 1, ..., 9, A, B, ..., Z podent treballar així fins amb bases fins a la base 36. El procediment pot ser el següent:

1. Llegim les dues bases b_1 i b_2 .
2. Llegim el nombre en base b_1 convertint-lo a base 10 mitjançant el mètode de les potències successives.
3. Convertim el nombre a base b_2 mitjançant el mètode de les divisions successives.

Caldrà saber treballar amb strings que, de fet, són vectors de caràcters.

Programa en Pascal

```
PROGRAM A7_13;

CONST MAX_BASE = 36;           (* Base més gran *)
      MAX_DEC = 10;           (* Nombre màxim de decimals *)

VAR xifres : STRING[MAX_BASE]; (* Xifres de totes les bases *)
    nombre1 : STRING;         (* Nombre a canviar de base *)
    nombre2 : STRING;         (* Nombre canviat de base *)
    nombre10int : LONGINT;     (* Nombre en base 10. Part dec *)
    nombre10frc : REAL;       (* Nombre en base 10. Part frc *)
    base1 : INTEGER;          (* Base inicial *)
    base2 : INTEGER;          (* Base final *)
    pos_punt : INTEGER;       (* Posició del punt decimal *)
    correcte : BOOLEAN;       (* Indica si nombre1 correcte *)
    i : INTEGER;              (* Comptador per bucles *)

BEGIN
  (* Inicialitzem les xifres de les bases *)
  xifres := '0123456789ABCDEFGHIJKLMNPOQRSTUVWXYZ';

  (* Llegim la primera base *)
  REPEAT
    WRITE('Introdueixi la primera base : ');
    READLN(base1);
  UNTIL (base1 >= 2) AND (base1 <= MAX_BASE);

  (* Llegim la segona base *)
  REPEAT
    WRITE('Introdueixi la segona base : ');
    READLN(base2);
  UNTIL (base2 >= 2) AND (base2 <= MAX_BASE);

  (* Llegim el nombre i comprovem que pertanyi a la primera base *)
  (* Al mateix temps canviem els valors ASCII de les xifres del *)
  (* nombre pel (codi ASCII del) seu valor decimal. *)
  REPEAT
    WRITE('Introdueixi el nombre a canviar de base : ');
    READLN(nombre1);

    i := 1;
    correcte := TRUE;
    pos_punt := LENGTH(nombre1) + 1;

    WHILE (i <= LENGTH(nombre1)) AND correcte DO
      BEGIN
        IF nombre1[i] = '.' THEN
          (* Comprovem que no tingui dos punts *)
          IF pos_punt <> LENGTH(nombre1) + 1 THEN
```

```
        correcte := FALSE
      ELSE
        pos_punt := i
      ELSE
        (* Busquem la xifra actual al vector de xifres *)
        BEGIN
          j := 1;
          nombre1[i] := UPCASE(nombre1[i]);
          WHILE (j <= base1) AND (nombre1[i] <> xifres[j]) DO
            j := j + 1;
          IF j <= base1 THEN
            nombre1[i] := CHR(j-1)
          ELSE
            correcte := FALSE;
          END;
          i := i + 1;
        END;
      UNTIL correcte;

      (* Passem el nombre de la primera base a base 10 *)

      (* Passem la part entera mitjançant potències successives *)
      nombre10int := 0;
      FOR i := 1 TO pos_punt-1 DO
        nombre10int := nombre10int*base1 + ORD(nombre1[i]);

      (* Passem la part fraccionaria mitjançant potències successives *)
      nombre10frc := 0;
      FOR i := LENGTH(nombre1) DOWNTO pos_punt+1 DO
        nombre10frc := (nombre10frc + ORD(nombre1[i])) / base1;

      (* Passem el nombre de base 10 a la segona base *)

      (* Passem la part entera mitjançant divisions successives *)
      nombre2 := '';
      WHILE nombre10int > 0 DO
        BEGIN
          nombre2 := xifres[(nombre10int MOD base2)+1] + nombre2;
          nombre10int := nombre10int DIV base2;
        END;

      (* Passem la part fraccionaria mitjançant multiplicacions successives
      *)
      IF nombre10frc <> 0 THEN
        BEGIN
          nombre2 := nombre2 + '.';
          i := 0;
          WHILE (nombre10frc <> 0) AND (i < MAX_DEC) DO
            BEGIN
              nombre2 := nombre2 + xifres[TRUNC(nombre10frc * base2)+1];
              nombre10frc := FRAC(nombre10frc * base2);
              i := i + 1;
            END;
          END;

      (* Imprimim el resultat *)
      WRITELN('El nombre en base ', base2, ' és ', nombre2);
    END.
```

Pràctica 8. Variables estructurades heterogènies

8.1 Dissenyeu un tipus de dades per representar cadascuna de les entitats següents:

- a) Un interval a la recta real.
- b) Un punt de l'espai.
- c) Una matriu en els reals de dimensions màximes 5x5.
- d) Un nombre complex.
- e) Una data.
- f) Una persona: nom, data de naixement i telèfon de contacte.
- g) Una agenda amb capacitat per guardar 100 persones.

Quan les dades que componen un nou tipus estructurat són totes del mateix tipus (per exemple, el punt de l'espai), quan és millor utilitzar un tipus estructurat homogeni (ARRAY) i quan un tipus estructurat heterogeni (RECORD)? **El principal avantatge de treballar amb variables estructurades homogènies és poder processar tots els seus elements de la mateixa manera mitjançant iteracions i accés indexat.** Quan no hem de fer servir aquesta característica, l'ús de variables estructurades heterogènies dóna més llegibilitat al programa. A l'exemple del punt de l'espai (sigui *a* una variable d'aquest tipus), per referenciar els valors de les seves coordenades és més elegant escriure *a.x*, *a.y* i *a.z* que *a[1]*, *a[2]* i *a[3]*, però si es tractés d'un punt d'un espai *n*-dimensional, on *n* és molt gran o introduït per l'usuari, i haguéssim de processar automàticament les coordenades, ens seria més útil accés indexat *a[1]*.

Programa en Pascal

```
PROGRAM A8_1;

TYPE

  (* a. Un interval a la recta real *)
  t_interval = RECORD
    limit_inf , limit_sup : REAL;
    obert_dret, obert_esq : BOOLEAN;
  END;

  (* b. Un punt de l'espai *)
  t_punt = RECORD
    x , y , z : REAL;
  END;

  (* c. Una matriu en els reals max 5x5 *)
  t_matriu = RECORD
    elements : ARRAY [1..5,1..5] OF REAL;
    fil, col : 1..5;
  END;
```

```
(* d. Un nombre complex *)
t_complex = RECORD
  real , imag : REAL;
END;

(* e. Una data *)
t_data = RECORD
  dia : 1..31;
  mes : 1..12;
  any : INTEGER;
END;

(* f. Una persona: nom, data naixement i telèfon *)
t_persona = RECORD
  nom      : STRING[30];
  naixement : t_data;
  telefon  : STRING[20];
END;

(* g. Agenda de 100 persones m...xim *)
t_agenda = RECORD
  persones : ARRAY [1..100] OF t_persona;
  num_pers : 1..100;
END;

BEGIN
END.
```

8.2 Volem construir una aplicació que ens permeti emmagatzemar les dades personals dels alumnes amb un màxim de 20. De cada un d'ells guardarem el nom, el cognom i les notes obtingudes en cinc pràctiques. El programa calcularà i guardarà la nota final com la mitjana de les cinc anteriors.

L'aplicació demanarà a l'usuari el nombre d'alumnes amb que vol treballar i les dades de cada un d'ells. A continuació, calcularà la nota final de tots els alumnes i visualitzarà el nom i la nota final de tots els alumnes.

Programa en Pascal

```
PROGRAM A8_2;

CONST MAX_AL = 20;
      MAX_NOTES = 5;

TYPE t_alumne = RECORD
  nom      : STRING;
  cognom   : STRING;
  notes    : ARRAY [1..MAX_NOTES] OF REAL;
  nota_total : REAL;
END;

t_classe = RECORD
  alumnes : ARRAY [1..MAX_AL] OF t_alumne;
  num_al  : 1..MAX_AL;
END;

VAR classe : t_classe;
    i, j : INTEGER;
```

```

BEGIN

  (* Demanar el nombre d'alumnes i les seves dades *)
  REPEAT
    WRITE('Quants alumnes tenim (màxim ' , MAX_AL , ') ? ');
    READLN(classe.num_al);
  UNTIL (classe.num_al >= 1) AND (classe.num_al <= MAX_AL);

  FOR i := 1 TO classe.num_al DO
    WITH classe.alumnes[i] DO
      BEGIN
        WRITELN;
        WRITE('Introdueixi el nom de l''alumne ', i, ' : ');
        READLN(nom);
        WRITE('Introdueixi el cognom de l''alumne ', i, ' : ');
        READLN(cognom);
        FOR j := 1 TO MAX_NOTES DO
          REPEAT
            WRITE('Introdueixi la nota ', j, ' de l''alumne ', i, ' : ');
          );
          READLN(notes[j]);
          UNTIL (notes[j] >= 0) AND (notes[j] <= 10);
        END;

      (* Calcular la nota final dels alumnes *)
      FOR i := 1 TO classe.num_al DO
        WITH classe.alumnes[i] DO
          BEGIN
            nota_total := 0;
            FOR j := 1 TO MAX_NOTES DO
              nota_total := nota_total + notes[j];
            nota_total := nota_total / MAX_NOTES;
          END;

          (* Escriure el resultat de la nota final de cada alumne *)
          WRITELN;
          WRITELN('Informe de notes finals');
          WRITELN('-----');
          FOR i := 1 TO classe.num_al DO
            WITH classe.alumnes[i] DO
              WRITELN(nom , ' ', cognom, ' : ', nota_total:4:2);
            END;
          END.

```

8.3 Amplieu l'exercici 7.6 per tal que permeti fer multiplicacions de matrius no quadrades, utilitzant per emmagatzemar la informació de cada matriu un registre que contindrà els següents camps:

- Una matriu de 10×10 elements.
- Una variable que indiqui el nombre de files reals de la matriu.
- Una variable que indiqui el nombre de columnes reals de la matriu.

```

TYPE t_matriu = RECORD
  valors : ARRAY [1..10,1..10] OF REAL;
  fil : 1..10;
  col : 1..10;
END;

```

Les matrius poden tenir un nombre de files diferent al de les columnes, amb una dimensió màxima de 10 files per 10 columnes. Tingueu present, però, que per poder multiplicar dues matrius, A i B , s'ha de complir la següent condició:

nombre de columnes d' A = nombre de files de B

i que la matriu resultant, C , compleix:

nombre de files de C = nombre de files d' A

nombre de columnes de C = nombre de columnes de B

Programa en Pascal

```
PROGRAM A8_3;

CONST
  MAX_FIL = 10;          (* Màxim nombre de files de la matriu *)
  MAX_COL = 10;         (* Màxim nombre de columnes de la matriu *)

TYPE t_matriu = RECORD
  valors   : ARRAY [ 1..MAX_FIL , 1..MAX_COL ] OF REAL;
  files    : 1..MAX_FIL;
  columnes : 1..MAX_COL;
END;

VAR
  i , j , k : INTEGER;
  M1 , M2 , P : t_matriu;

BEGIN
  WRITELN('Multiplicació de dues matrius de qualsevol dimensió');
  WRITELN('-----');

  (* Demanem el nombre de files i columnes de la primera matriu *)
  WRITELN;
  REPEAT
    WRITE('Introdueixi les files i columnes de la 1ª matriu : ');
    READLN( M1.files , M1.columnes );
  UNTIL (M1.files >= 1) AND (M1.files <= MAX_FIL) AND
        (M1.columnes >= 1) AND (M1.columnes <= MAX_COL);

  (* Omplim la primera matriu *)
  WRITELN;
  FOR i := 1 TO M1.files DO
    FOR j := 1 TO M1.columnes DO
      BEGIN
        WRITE('Introdueixi l'element [ ' , i , ' , ' , j ,
              ' ] de la 1ª matriu : ');
        READLN(M1.valors[i,j]);
      END;
    END;

  (* Demanem el nombre de files i columnes de la segona matriu *)
  WRITELN;
  REPEAT
    WRITE('Introdueixi les files i columnes de la 2ª matriu : ');
    READLN( M2.files , M2.columnes );
  UNTIL (M2.files = M1.columnes) AND
        (M2.columnes >= 1) AND (M2.columnes <= MAX_COL);

  (* Omplim la segona matriu *)
  WRITELN;
  FOR i := 1 TO M2.files DO
    FOR j := 1 TO M2.columnes DO
```

```

BEGIN
WRITE('Introdueixi l'element [' , i , ',' , j ,
      ' ] de la 2ª matriu : ' );
READLN(M2.valors[i,j]);
END;

(* Calculem la matriu producte *)
P.files      := M1.files;
P.columnes   := M2.columnes;
FOR i := 1 TO P.files DO
  FOR j := 1 TO P.columnes DO
    BEGIN
      P.valors[i,j] := 0;
      FOR k := 1 TO M1.columnes DO
        P.valors[i,j] := P.valors[i,j] +
                        M1.valors[i,k]*M2.valors[k,j];
      END;
    END;
  END;

(* Imprimim la matriu producte *)
WRITELN;
WRITELN('La matriu producte és:');
FOR i := 1 TO P.files DO
  BEGIN
    WRITE('[');
    FOR j := 1 TO P.columnes DO WRITE(P.valors[i,j]:9:2);
    WRITELN(' ]');
  END;
END.

```

8.4 Amplieu l'exercici 7.4 per tal que permeti trobar el centre de gravetat d'un conjunt de punts de l'espai bidimensional emmagatzemats en un vector.

Un punt estarà format per un nom que l'identifica i les seves coordenades:

t_punt

id	x	y
----	---	---

Programa en Pascal

```

PROGRAM A8_4;

CONST MaxElements = 10;

TYPE
  t_id_punt = STRING[30];

  t_punt = RECORD
    id      : t_id_punt;
    x , y  : REAL;
  END;

  t_vector_punts = RECORD
    elements : ARRAY [1..MaxElements] OF t_punt;
    num_elem : 0..MaxElements;
  END;

VAR
  punts : t_vector_punts;
  posicio : 1..MaxElements;
  m : t_punt;

BEGIN

```

```

WRITELN('Centre de gravetat dels punts guardats en un vector');
WRITELN('-----');

(* Llegim quants punts vol introduir *)
REPEAT
  WRITE('Quants punts vols introduir (de 1 a ', MaxElements, ')? ');
  READLN(punts.num_elem);
UNTIL (punts.num_elem >= 1) AND (punts.num_elem <= MaxElements);

(* Llegim els punts *)
FOR posicio := 1 TO punts.num_elem DO
  BEGIN
    WRITE('Introdueix el nom del punt ', posicio, ' : ');
    READLN(punts.elements[posicio].id);
    WRITE('Introdueix les coordenades del punt ', posicio, ' : ');
    READLN(punts.elements[posicio].x);
    READLN(punts.elements[posicio].y);
  END;

(* Calculem el punt mig *)
m.id := 'centre de gravetat';
m.x := 0;
m.y := 0;
FOR posicio := 1 TO punts.num_elem DO
  BEGIN
    m.x := m.x + punts.elements[posicio].x;
    m.y := m.y + punts.elements[posicio].y;
  END;
m.x := m.x / punts.num_elem;
m.y := m.y / punts.num_elem;

(* Mostrem el punt mig *)
WRITELN(m.id, ' : (', m.x:4:2, ', ', m.y:4:2, ')');
END.

```

8.5 Dissenyeu un algorisme en pseudocodi i el corresponent programa en Pascal que ens permeti sumar i multiplicar dos polinomis. Els polinomis els implementarem com un registre on guardarem:

- | El grau del polinomi. Limitarem el grau màxim a 10.
- | Els coeficients del polinomi.

t_polinomi	grau	coeficients										
		0	1	2	3	4	5	6	7	8	9	10

Cal recordar les normes de suma i multiplicació de polinomis. Es recomana fer a mà un exemple de cada per veure la mecànica de treball, per exemple:

Polinomi 1: $2x^3 + 3x^2 - 2$	3	-2	0	3	2							
Polinomi 2: $-2x^2 + x - 3$	2	-3	1	-2								
Suma: $2x^3 + x^2 + x - 5$	3	-5	1	1	2							
Mult: $-4x^5 - 4x^4 - 3x^3 - 5x^2 - 2x + 6$	5	6	-2	-5	-3	-4	-4					

Programa en Pascal

```
PROGRAM A8_5;

CONST MAX_GRAU = 10;

TYPE t_polinomi = RECORD
    grau : 0..MAX_GRAU;
    coef : ARRAY [0..MAX_GRAU] OF REAL;
END;

VAR p1 , p2 , p3 : t_polinomi;
    i , j : INTEGER;

BEGIN
    WRITELN('Suma i multiplicació de polinomis');
    WRITELN('-----');

    (* Llegim el grau del primer polinomi *)
    WRITELN;
    REPEAT
        WRITE('Introdueixi el grau del primer polinomi : ');
        READLN( p1.grau );
    UNTIL (p1.grau >= 0) AND (p1.grau <= MAX_GRAU);

    (* Llegim els coeficients del primer polinomi. *)
    (* El coeficient de grau màxim no pot ser zero. *)
    REPEAT
        WRITE('Introdueixi el coeficient del terme de grau ', p1.grau,
            ' : ');
        READLN(p1.coef[p1.grau]);
    UNTIL (p1.coef[p1.grau] <> 0) OR (p1.grau = 0);
    FOR i := p1.grau-1 DOWNTO 0 DO
        BEGIN
            WRITE('Introdueixi el coeficient del terme de grau ', i, ' : ');
            READLN(p1.coef[i]);
        END;

    (* Llegim el grau del segon polinomi *)
    WRITELN;
    REPEAT
        WRITE('Introdueixi el grau del segon polinomi : ');
        READLN( p2.grau );
    UNTIL (p2.grau >= 0) AND (p2.grau <= MAX_GRAU);

    (* Llegim els coeficients del segon polinomi. *)
    (* El coeficient de grau màxim no pot ser zero. *)
    REPEAT
        WRITE('Introdueixi el coeficient del terme de grau ', p2.grau,
            ' : ');
        READLN(p2.coef[p2.grau]);
    UNTIL (p2.coef[p2.grau] <> 0) OR (p2.grau = 0);
    FOR i := p2.grau-1 DOWNTO 0 DO
        BEGIN
            WRITE('Introdueixi el coeficient del terme de grau ', i, ' : ');
            READLN(p2.coef[i]);
        END;

    (* Calculem la suma de polinomis *)
    IF p1.grau > p2.grau THEN
        BEGIN
            p3.grau := p1.grau;
            FOR i := 0 TO p2.grau DO p3.coef[i] := p1.coef[i] + p2.coef[i];
            FOR i := p2.grau+1 TO p1.grau DO p3.coef[i] := p1.coef[i];
        END
    ELSE
        BEGIN
            p3.grau := p2.grau;
```

```
FOR i := 0 TO p1.grau DO p3.coef[i] := p1.coef[i] + p2.coef[i];
FOR i := p1.grau+1 TO p2.grau DO p3.coef[i] := p2.coef[i];
END;
(* Actualitzem el grau del polinomi resultant *)
WHILE (p3.coef[p3.grau] = 0) AND (p3.grau > 0) DO
  p3.grau := p3.grau - 1;

(* Imprimim el polinomi resultat *)
WRITELN;
WRITE('Suma :');
FOR i := p3.grau DOWNTO 0 DO
  IF p3.coef[i] <> 0 THEN
    BEGIN

      (* Escribim el signe *)
      IF p3.coef[i] > 0 THEN
        WRITE(' + ')
      ELSE
        WRITE(' - ');

      (* Escribim el coeficient *)
      WRITE(ABS(p3.coef[i]):4:2);

      (* Escribim l'exponent *)
      IF i > 1 THEN
        WRITE(' x^', i)
      ELSE
        IF i = 1 THEN
          WRITE(' x')
        END
      ELSE
        (* Comprovem cas especial polinomi = 0 *)
        IF p3.grau = 0 THEN
          WRITE(0);
        END
      END
    END
  END
WRITELN;

(* Calculem la multiplicació de polinomis *)
IF p1.grau + p2.grau > MAX_GRAU THEN
  WRITELN('Multiplicació : el grau del polinomi resultat, ',
    p1.grau + p2.grau, ', sobrepassa el grau màxim, ',
    MAX_GRAU)
ELSE
  BEGIN
    p3.grau := p1.grau + p2.grau;
    FOR i := 0 TO p3.grau DO
      p3.coef[i] := 0;
    FOR i := 0 TO p1.grau DO
      FOR j := 0 TO p2.grau DO
        p3.coef[i+j] := p3.coef[i+j] + p1.coef[i]*p2.coef[j];
        (* Es pot fer el mateix potser més clar en tres línies, *)
        (* si tenim definides les variables nou_grau i nou_coef *)
        (* nou_grau := i + j; *)
        (* nou_coef := p1.coef[i]*p2.coef[j]; *)
        (* p3.coef[nou_grau] := p3.coef[nou_grau] + nou_coef *)
      END
    END
  END

  (* Imprimim el polinomi resultat *)
  WRITELN;
  WRITE('Multiplicació :');
  FOR i := p3.grau DOWNTO 0 DO
    IF p3.coef[i] <> 0 THEN
      BEGIN

        (* Escribim el signe *)
        IF p3.coef[i] > 0 THEN
          WRITE(' + ')
        ELSE
          WRITE(' - ');
```

```
(* Escribim el coeficient *)  
WRITE(ABS(p3.coef[i]):4:2);  
  
(* Escribim l'exponent *)  
IF i > 1 THEN  
    WRITE(' x^', i)  
ELSE  
    IF i = 1 THEN  
        WRITE(' x')  
    END  
ELSE  
    (* Comprovem cas especial polinomi = 0 *)  
    IF p3.grau = 0 THEN  
        WRITE(0);  
    END  
WRITELN;  
END;  
END.
```

Pràctica 9. Disseny d'una aplicació. Subprogrames i mòduls.

9.1 Observeu el programa i responeu les següents qüestions:

- Quin és el valor de les variables x i y després de cridar al procediment *prova*? $x=25$ i $y=4$.
- Quina diferència hi ha entre pas de paràmetres per valor i per referència? **Al pas de paràmetres per valor (altrament anomenat d'entrada i protegit): (1) l'argument que passem pot ser una constant, variable o expressió del mateix tipus que el paràmetre; (2) el paràmetre és una variable local on es copia el valor de l'argument; (3) i per tant, per molt que modifiquem el valor del paràmetre, no es modifica el valor de l'argument amb que es va cridar el subprograma. Al pas de paràmetres per referència (altrament anomenat de sortida i desprotegit): (1) l'argument que passem només pot ser una variable del mateix tipus que el paràmetre; (2) el paràmetre és un "segon nom" que fa referència a la mateixa variable de l'argument; (3) i per tant, quan modifiquem el valor del paràmetre, es modifica el valor de l'argument amb que es va cridar el subprograma. Per exemple: en el pas de paràmetres de la crida del nostre programa trobem que $n1$ és per valor ("variable a part") i quan modifiquem $n1$ no modifiquem x , mentre que $n2$ és per referència ("segon nom") i quan modifiquem $n2$ modifiquem y .**

x $n1$ | y $n2$

Quin seria el valor de les variables x i y després de cridar al procediment *prova* si canviéssim el seu encapçalament per `PROCEDURE prova(VAR n1: INTEGER; n2: INTEGER)? $x=20$ i $y=20$.`

- Per què és convenient que l'entrada i sortida a un subprograma es realitzi mitjançant paràmetres d'entrada, paràmetres de sortida i el valor de retorn (si es tracta d'una funció), i no mitjançant `READ()` i `WRITE()`? **Ens proporciona flexibilitat. No sempre haurem de llegir les dades d'entrada (de vegades les haurem de calcular). No sempre haurem d'imprimir les dades de sortida (de vegades seran resultats parcials). Deixem que el programador/a que utilitza la funció o procediment decideixi que vol fer amb aquestes dades. Imaginem quina catàstrofe si la funció `sinus` demanés sempre l'angle pel teclat i imprimís el resultat per pantalla. Què passaria llavors amb la instrucció `c:= a*b*sin(x)*sin(y)`?**
- Per què no és convenient fer servir variables globals dins una funció o procediment? **Perquè es perd la modularitat. Una funció o procediment ha**

de ser una “caixa negra”, que rep les dades d’entrada mitjançant uns arguments, realitza operacions únicament en funció dels valors d’aquests, i retorna una o més dades de sortida. Als mateixos paràmetres d’entrada correspondrà sempre la mateixa sortida. (1) És més senzill localitzar errors a un programa si els subprogrames funcionen sempre de la mateixa manera, sense dependre de valors de variables externes. (2) És més senzill localitzar errors a un programa si cap variable del programa es modifica de manera amagada per un subprograma. (3) No es pot exportar un subprograma d’un programa a un altre si aquest no funciona únicament en base als seus paràmetres i variables locals. Si funciona utilitzant variables globals, fallarà en emprar-les en un nou programa que no les tingui definides o que en faci un ús diferent. Quin seria el valor de les variables x i y després de cridar al procediment *prova* si no existís dins seu la declaració local `VAR x: INTEGER; x=2 i y=4`.

Programa en Pascal

```
PROGRAM A9_1;

VAR x, y: INTEGER;

PROCEDURE prova(n1:INTEGER; VAR n2:INTEGER);
VAR x: INTEGER;
BEGIN
  x := n2;
  y := 5*x*x;
  n1 := y;
  n2 := 4;
END;

BEGIN
  x := 25;
  y := 2;
  WRITELN('Abans      x = ', x, '      y = ', y);
  prova(x, y);
  WRITELN('Després    x = ', x, '      y = ', y);
END.
```

9.2 Escriviu els subprogrames corresponents a les següents especificacions, considerant a cada cas si és més adient emprar funció o procediment.

- a) Un subprograma tal que, donats dos reals a i b , calcula el logaritme en base b d' a segons la següent fórmula: $\log_b a = (\ln a) / (\ln b)$.

```
FUNCTION Log(a, b : REAL) : REAL;
BEGIN
  Log := LN(a)/LN(b);
END;
```

- b) Un subprograma tal que, donat un enter, calcula el signe de l'enter i ens torna -1, 0 1 segons si l'enter és negatiu, zero o positiu.

```
FUNCTION Signe(n : INTEGER) : INTEGER;
BEGIN
  IF n > 0 THEN
    Signe := 1
  ELSE
    IF n < 0 THEN
      Signe := -1
    ELSE
      Signe := 0;
  END;
END;
```

- c) Un subprograma tal que, donat un caràcter, ens calcula si el caràcter en qüestió és una lletra o no.

```
FUNCTION Lletra(c : CHAR) : BOOLEAN;
BEGIN
  Lletra := c IN ['A'..'Z','a'..'z'];

  (* També podem escriure *)
  (* Lletra := ((c >= 'a') AND (c <= 'z')) OR ((c >= 'A') AND (c <= 'Z')); *)

  (* Però de la primera manera és més senzill considerar altres lletres: *)
  (* Lletra := c IN ['A'..'Z','À','È','É','Í','Ò','Ó','Ú','Û','Ç','Ñ', etc.*)
END;
```

- d) Un subprograma tal que, donats tres reals, obté el mínim interval tancat que conté tots tres.

```
PROCEDURE interval(a, b, c : REAL; VAR min, max : REAL);
BEGIN
  min := a;
  max := a;

  IF b > max THEN
    max := b
  ELSE
    min := b;

  IF c > max THEN
    max := c
  ELSE
    IF c < min THEN
      min := c;
  END;
END;
```

- e) Un subprograma tal que, donats dos vectors de 10 elements, retorna el vector resultat de sumar els dos anteriors.

```
CONST NUM_EL = 10;

TYPE t_vector = ARRAY [1..NUM_EL] OF REAL;

PROCEDURE SumaVect(VAR vector1, vector2, vector_suma : t_vector);
VAR i : INTEGER;
BEGIN
  FOR i := 1 TO NUM_EL DO
    vector_suma[i] := vector1[i] + vector2[i];
  END;
END;
```

Hi ha una altre manera de fer-ho que ens permet sumar vectors de qualsevol mida. Es tracta de passar vectors oberts com a paràmetres de la funció o procediment i posteriorment fer servir les funcions LOW i HIGH de Pascal per trobar els límits dels vectors.

```
PROCEDURE SumaVect(VAR vector1, vector2, vector_suma : ARRAY OF REAL);
VAR i : INTEGER;
BEGIN
  FOR i := LOW(vector_suma) TO HIGH(vector_suma) DO
    vector_suma[i] := vector1[i] + vector2[i];
  END;
END;
```

- f) Un subprograma tal que, donat un polinomi, retorna la seva derivada. Feu servir el tipus polinomi definit a l'exercici 8.5.

```
CONST MAX_GRAU = 10;

TYPE t_polinomi = RECORD
  grau : 0..MAX_GRAU;
  coef : ARRAY [0..MAX_GRAU] OF REAL;
END;

PROCEDURE DerivPoli(VAR p : t_polinomi);
VAR i : INTEGER;
BEGIN
  IF p.grau = 0 THEN
    p.coef[0] := 0
  ELSE
    BEGIN
      FOR i := 0 TO p.grau-1 DO
        p.coef[i] := (i+1)*p.coef[i+1];
      p.grau := p.grau - 1;
    END;
  END;
END;
```

- g) Un subprograma tal que, donat un enter, calcula recursivament el seu factorial (aquest nombre enter multiplicat per tots els seus antecessors):

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1 = n \times (n-1)! \text{ si } n > 1 \quad \text{i} \quad 1 \text{ si } n=1 \text{ ó } n=0$$

```
FUNCTION Factorial(n : LONGINT):LONGINT;
BEGIN
  IF (n = 0) OR (n = 1) THEN
    Factorial := 1
  ELSE

```

```
Factorial := n*Factorial(n-1);  
END;
```

Quina diferència hi ha entre funció i procediment? A Pascal, una funció correspon al concepte matemàtic de funció, mentre que un procediment correspon al concepte d'instrucció. En retornar d'una crida a una funció, el cos de la crida queda substituït per un valor que es pot emprar dintre d'una expressió o instrucció (per exemple, `a := 2*sin(x)`) mentre que a un procediment això no passa (per exemple, no té cap sentit `a := 2*write(x)`).

Com es pot convertir una funció en procediment i viceversa? Tota funció es pot convertir en procediment: (1) canviant la paraula `FUNCTION` per `PROCEDURE`, (2) convertint el tipus de retorn de la funció en un paràmetre addicional passat per referència on es retornarà l'antiga sortida de la funció i (3) canviant totes les crides a l'antiga funció per tal de convertir-les en instruccions i que incloguin el nou paràmetre. Per exemple:

<pre>FUNCTION sin(x:REAL):REAL; VAR declaració var. locals BEGIN instruccions; sin := resultat; END; ... area := a*b*sin(alfa);</pre>	<pre>PROCEDURE sin(x:REAL; VAR sortida:REAL); VAR declaració var. locals BEGIN instruccions; sortida := resultat; END; ... sin(alfa, r); area := a*b*r;</pre>
---	---

Tot procediment amb un únic paràmetre de sortida es pot convertir en funció seguint de manera inversa els mateixos passos.

9.3 Realitzeu el disseny modular de l'exercici 8.3 (multiplicació de matrius). La definició de les matrius és la mateixa. Només cal definir tres subprogrames:

- Un procediment que llegeixi per teclat els elements d'una estructura de tipus `t_matriu`, passada com a paràmetre:

```
PROCEDURE LlegirMatriu(VAR M : t_matriu);
```

- Un procediment per visualitzar per pantalla els elements d'una estructura de tipus `t_matriu`, passada com a paràmetre:

```
PROCEDURE VeureMatriu(VAR M : t_matriu);
```

- Una funció que calculi el producte de dues matrius passades com a paràmetre, i deixi el resultat en una tercera matriu també passada com a paràmetre, en el cas que això sigui possible.

```
FUNCTION MultMatrius(VAR M1, M2, R : t_matriu):BOOLEAN;
```

La funció per multiplicar matrius intentarà calcular $R = M_1 \times M_2$ i, si el càlcul és possible, ens retornarà en R el valor de la matriu producte i la funció retornarà el valor booleà TRUE. Si el càlcul no és possible la funció retornarà el valor booleà FALSE.

Programa en Pascal

```

PROGRAM A9_3;

CONST
  MAX_FIL = 10;          (* Màxim nombre de files de la matriu *)
  MAX_COL = 10;         (* Màxim nombre de columnes de la matriu *)

TYPE t_matriu = RECORD
  valors   : ARRAY [ 1..MAX_FIL , 1..MAX_COL ] OF REAL;
  files    : 1..MAX_FIL;
  columnes : 1..MAX_COL;
END;

VAR
  M1, M2, P : t_matriu;

(* LlegirMatriu: *)
(* Procediment que llegeix per teclat els elements *)
(* d'una matriu passada com a paràmetre. *)
PROCEDURE LlegirMatriu(VAR M : t_matriu);
VAR i, j : INTEGER;
BEGIN
  (* Demanem el nombre de files i columnes de la matriu *)
  WRITELN;
  REPEAT
    WRITE('Introdueixi el nombre de files i columnes de la matriu : ');
    READLN( M.files , M.columnes );
  UNTIL (M.files >= 1) AND (M.files <= MAX_FIL) AND
        (M.columnes >= 1) AND (M.columnes <= MAX_COL);

  (* Demanem els elements de la matriu *)
  WRITELN;
  FOR i := 1 TO M.files DO
    FOR j := 1 TO M.columnes DO
      BEGIN
        WRITE('Introdueixi l'element [' , i , ', ' , j , '] de la matriu : ');
      );
      READLN(M.valors[i,j]);
    END;
  END;
END;

(* MultMatrius: *)
(* Funció que calcula el producte de dues matrius passades com a *)
(* paràmetre i deixa el resultat en una tercera matriu, també passada *)
(* com a paràmetre, en el cas que això sigui possible. *)
(* La funció retorna el valor booleà TRUE si el càlcul és possible, i *)
(* el valor booleà FALSE si no ho és. *)
FUNCTION MultMatrius(VAR M1, M2, R : t_matriu):BOOLEAN;
VAR i, j, k : INTEGER;
BEGIN
  (* Comprobem que sigui possible la multiplicació *)
  IF M1.columnes <> M2.files THEN
    MultMatrius := FALSE
  ELSE
    BEGIN

```

```

    (* Calculem la matriu producte *)
    R.files := M1.files;
    R.columns := M2.columns;
    FOR i := 1 TO R.files DO
        FOR j := 1 TO R.columns DO
            BEGIN
                R.valors[i,j] := 0;
                FOR k := 1 TO M1.columns DO
                    R.valors[i,j] :=
R.valors[i,j]+M1.valors[i,k]*M2.valors[k,j];
                END;
                MultMatrius := TRUE;
            END;
        END;

    (* VeureMatriu: *)
    (* Procediment que visualitza per pantalla els elements *)
    (* d'una matriu passada com a paràmetre. *)
    PROCEDURE VeureMatriu(VAR M : t_matriu);
    VAR i, j : INTEGER;
    BEGIN
        FOR i := 1 TO M.files DO
            BEGIN
                WRITE(' ');
                FOR j := 1 TO M.columns DO WRITE(M.valors[i,j]:9:2);
                WRITELN(' ');
            END;
        END;

    (* Programa principal *)
    BEGIN
        WRITELN('Multiplicació de dues matrius de qualsevol dimensió');
        WRITELN('-----');

        (* Llegim les matrius *)
        LlegirMatriu(M1);
        LlegirMatriu(M2);

        (* Calculem i imprimim la matriu producte *)
        WRITELN;
        IF MultMatrius(M1, M2, P) THEN
            BEGIN
                WRITELN('La matriu producte és:');
                VeureMatriu(P);
            END
        ELSE
            WRITELN('Dimensions incorrectes');
        END.
    
```

9.4 Amplieu l'exercici 8.4 per tal que permeti introduir noms i coordenades de punts de l'espai bidimensional i posteriorment calcular el centre de gravetat d'aquests punts.

Un punt estarà format per un nom que l'identifica i les seves coordenades:

t_punt

id	x	y
----	---	---

En iniciar el programa les opcions que ens apareixeran seran:

1. Introduir punts.

2. Visualitzar punts.
3. Modificar un punt cercant pel nom.
4. Calcular el centre de gravetat.
5. Finalitzar.

El disseny de l'algorisme ha de ser modular. Feu servir funcions i subrutines per implementar les diferents opcions del menú.

Programa en PASCAL

```
PROGRAM A9_4;

USES crt;

CONST MaxElements = 5;

TYPE
  t_id_punt = STRING[15];

  t_punt = RECORD
    id   : t_id_punt;
    x , y : REAL;
  END;

  t_vector_punts = RECORD
    punts      : ARRAY [1..MaxElements] OF t_punt;
    num_punts  : 0..MaxElements;
  END;

VAR
  v_punts      : t_vector_punts;
  centre_grav : t_punt;
  opcio        : 1..5;

(* IntroduirPunts : *)
(* Procediment que demana punts per pantalla i *)
(* els guarda al vector passat com a paràmetre. *)
PROCEDURE IntroduirPunts(VAR vector : t_vector_punts);
VAR punt : t_punt;
BEGIN
  (* Demana noms i coordenades i els guarda al vector *)
  (* mentre no s'introdueix ENTER en el camp del nom *)
  (* i tampoc s'arribi al final del vector. *)
  IF vector.num_punts < MaxElements THEN
    BEGIN
      WRITE('Introdueix el nom del punt o ENTER per acabar : ');
      READLN(punt.id);
      END;
    WHILE (punt.id <> '') AND (vector.num_punts < MaxElements) DO
      BEGIN
        WRITE('Introdueix les coordenades corresponents a aquest punt:');
        READLN(punt.x, punt.y);
        vector.num_punts := vector.num_punts + 1;
        vector.punts[vector.num_punts] := punt;
        IF vector.num_punts < MaxElements THEN
          BEGIN
            WRITE('Introdueix el nom del punt o ENTER per acabar : ');
            READLN(punt.id);
            END;
          END;
    END;
END;
```

```
(* LListarPunts : *)
(* Procediment que llista per pantalla tots els punts *)
(* del vector passat com a paràmetre. *)
PROCEDURE LlistarPunts(VAR vector : t_vector_punts);
VAR i : INTEGER;
BEGIN
  FOR i := 1 TO vector.num_punts DO
    WRITELN(vector.punts[i].id, ' : (',
            vector.punts[i].x:4:2, ', ',
            vector.punts[i].y:4:2, ')');
  END;

(* ModificarPunt : *)
(* Procediment que demana l'identificador d'un punt i el busca al *)
(* vector passat com a paràmetre. Quan el troba, demana les noves *)
(* coordenades i les torna a guardar al vector. *)
PROCEDURE ModificarPunt(VAR vector : t_vector_punts);
VAR id_punt : t_id_punt;
    i : INTEGER;
BEGIN
  IF vector.num_punts > 0 THEN
    BEGIN
      (* Llegeix els punts un a un fins trobar el punt buscat. *)
      WRITE('Introdueixi l''identificador del punt a buscar : ');
      READLN(id_punt);
      i := 1;
      WHILE (i<= vector.num_punts) AND (id_punt <> vector.punts[i].id) DO
        i := i + 1;

      (* Si l'ha trobat l'escriu per pantalla i demana les coordenades *)
      IF id_punt = vector.punts[i].id THEN
        BEGIN
          WRITELN(vector.punts[i].id, ' : (',
                  vector.punts[i].x:4:2, ', ',
                  vector.punts[i].y:4:2, ')');
          WRITE('Introdueix les noves coordenades d'aquest punt : ');
          READLN(vector.punts[i].x, vector.punts[i].y);
        END
      ELSE
        WRITELN('Punt no trobat.');
```

```
      END;
    END;

(* CalcularCentreGrav : *)
(* Funció que calcula el centre de gravetat dels punts del vector *)
(* passat com a paràmetre. Si pot calcular el centre de gravetat, *)
(* el retorna a la variable 'centre_grav' i la funció retorna TRUE. *)
(* Si no pot calcular el centre de gravetat la funció retorna FALSE. *)
FUNCTION CalcularCentreGrav(VAR vector:t_vector_punts; VAR
centre_grav:t_punt):BOOLEAN;
VAR punt : t_punt;
    i : INTEGER;
BEGIN
  (* Processa els punts un a un, calculant el centre de gravetat. *)
  IF vector.num_punts > 0 THEN
    BEGIN
      centre_grav.id := 'Centre grav';
      centre_grav.x := 0;
      centre_grav.y := 0;
      FOR i := 1 TO vector.num_punts DO
        BEGIN
          centre_grav.x := centre_grav.x + vector.punts[i].x;
          centre_grav.y := centre_grav.y + vector.punts[i].y;
        END;
      centre_grav.x := centre_grav.x / vector.num_punts;
```

```
        centre_grav.y := centre_grav.y / vector.num_punts;
        CalcularCentreGrav := TRUE;
    END
ELSE
    CalcularCentreGrav := FALSE;
END;

BEGIN (* Programa principal *)

    v_punts.num_punts := 0;

    REPEAT
        CLRSCR;
        WRITELN('Menú');
        WRITELN('-----');
        WRITELN('1. Introduir punts           ');
        WRITELN('2. Llistar punts           ');
        WRITELN('3. Modificar un punt      ');
        WRITELN('4. Calcular el centre de gravetat');
        WRITELN('5. Finalitzar            ');
        WRITELN('-----');
        WRITE('Opció : ');
        READLN(opcio);
        WRITELN;

        CASE opcio OF

            1 : IntroduirPunts(v_punts);

            2 : LlistarPunts(v_punts);

            3 : ModificarPunt(v_punts);

            4 : IF CalcularCentreGrav(v_punts, centre_grav) THEN
                    WRITELN('El centre de gravetat és (',
                            centre_grav.x:4:2, ',',
                            centre_grav.y:4:2, ')');
                ELSE
                    WRITELN('No s''ha introduït cap punt');
                END IF;

            5 : WRITELN('Gràcies per utilitzar CGRAV v 1.0 !');

        ELSE
            WRITELN('Opció incorrecta');
        END;

        WRITE('Premi <ENTER> per continuar ');
        READLN;

    UNTIL opcio = 5;

END.
```

9.5 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal tal que ens permetin realitzar la resolució de sistemes d'equacions lineals 3×3 . Per resoldre aquests sistemes utilitzarem el mètode de Kramer. Aquest mètode utilitza el càlcul de determinants, que es pot realitzar mitjançant la utilització de la fórmula de càlcul de determinants 3×3 .

Aprofiteu l'algorisme de l'exercici 7.5 per resoldre l'exercici.

Quina és la millor manera de passar una matriu com a paràmetre d'un subprograma: per valor o per referència? **Per referència.** Per què? **La crida a la funció és més ràpida i ens estalviem memòria, ja que no s'han de copiar tots els valors de la matriu sinó només una referència a aquesta. Per contra, hem d'anar amb compte de no modificar els elements de la matriu de manera no desitjada, doncs en retornar de la crida a la funció els valors d'aquests elements hauran variat.**

Programa en Pascal

```
PROGRAM A9_5;

TYPE
  t_matriu3x3 = ARRAY[1..3 , 1..3] OF REAL;
  t_matriu3x4 = ARRAY[1..3 , 1..4] OF REAL;

VAR
  i , j , k : INTEGER;
  matriu_ext : t_matriu3x4;
  matriu_aux : t_matriu3x3;
  vector_sol : ARRAY [1..4] OF REAL;

(* det3x3 : *)
(* funció que calcula el determinant d'una matriu 3x3 *)
FUNCTION det3x3(VAR m : t_matriu3x3):REAL;
BEGIN
  det3x3 := m[1,1]*m[2,2]*m[3,3] + m[1,2]*m[2,3]*m[3,1]
           + m[1,3]*m[2,1]*m[3,2] - m[1,1]*m[2,3]*m[3,2]
           - m[1,2]*m[2,1]*m[3,3] - m[1,3]*m[2,2]*m[3,1];
END;

BEGIN
  WRITELN('Resolució d'equacions lineals 3 x 3');
  WRITELN('-----');

  (* Omplim la matriu *)
  WRITELN;
  FOR i := 1 TO 3 DO
    FOR j := 1 TO 4 DO
      BEGIN
        WRITE('Introdueixi l'element [' , i , ',' , j ,
              '] de la matriu : ');
        READLN(matriu_ext[i,j]);
      END;
    END;

  (* Calculem el determinant de les matrius 3x3 resultants de *)
  (* substituir cadascuna de les columnes de la matriu estesa 3x4 *)
  (* per la columna solució. Amb k assenyalaré la columna, i *)
  (* guardaré la solució a la posició k de vector_sol. *)
  FOR k := 1 TO 4 DO
    BEGIN
      (* Copio dins matriu_aux la matriu que queda de substituir la *)
      (* columna k a matriu_ext. Després trobo el seu determinant. *)
      FOR i := 1 TO 3 DO
        FOR j := 1 TO 3 DO
          IF j <> k THEN
            matriu_aux[i,j] := matriu_ext[i,j]
          ELSE
            matriu_aux[i,j] := 0;
          END;
        END;
      vector_sol[k] := det3x3(matriu_aux);
    END;
  END;
END;
```

```
        matriu_aux[i,j] := matriu_ext[i,4];
        vector_sol[k] := det3x3(matriu_aux);
        END;

        (* Imprimim el resultat *)
        WRITELN;
        IF vector_sol[4] = 0 THEN
            BEGIN
                WRITELN('El determinant és zero.');
```

```
                WRITELN('Sistema incompatible o amb infinites solucions.');
```

```
            END
        ELSE
            BEGIN
                WRITELN('x = ', vector_sol[1]/vector_sol[4]);
                WRITELN('y = ', vector_sol[2]/vector_sol[4]);
                WRITELN('z = ', vector_sol[3]/vector_sol[4]);
            END;
        END.
```

9.6 Redissenyu l'aplicació 5.10 utilitzant els criteris de programació modular. Per això implementareu els següents subprogrames en Pascal:

a) FUNCTION RangDada (min, max : INTEGER) : INTEGER ;

Funció que retorna una dada entrada per l'usuari pel teclat i que es troba dins dels límits marcats pels valors *min* i *max*. Feu servir l'algorisme de l'exercici 6.3.

b) FUNCTION DataOK (dia, mes, any : INTEGER) : BOOLEAN ;

Funció que donats tres valors enters que suposadament formen una data (dia-mes-any), retorna el valor TRUE si la data és correcta i FALSE en cas contrari. Feu servir l'algorisme de l'exercici 5.10.

c) PROCEDURE Pausa;

Procediment que genera una pausa en el programa. No se surt del procediment fins que no es premi la tecla <ENTER>.

El procediment RangDada serà utilitzat a l'hora de demanar el dia, mes i any; el procediment DataOK per comprovar si la data és correcta; i el procediment Pausa pot ser utilitzat a l'hora de parar la pantalla al final del programa.

Programa en Pascal

```
PROGRAM A9_6;

VAR dia, mes, any : INTEGER;

(* RangDada : *)
(* Funció que retorna una dada entrada per l'usuari pel teclat *)
(* i que es troba dins dels límits marcats pels valors max i min. *)
FUNCTION RangDada(min , max : INTEGER) : INTEGER;
VAR dada : INTEGER;
BEGIN
```

```
REPEAT
  WRITE('Introdueix una dada entera entre ',min,' i ',max,' : ');
  READLN(dada);
  IF (dada < min) OR (dada > max) THEN
    WRITELN('Valor incorrecte.');
```

```
UNTIL (dada >= min) AND (dada <= max);
RangDada := dada;
END;
```

```
(* DataOK : *)
(* Funció que donats tres valors enters que formen una data, retorna *)
(* el valor TRUE si la data és correcta i FALSE en cas contrari. *)
FUNCTION DataOK(dia , mes , any : INTEGER) : BOOLEAN;
BEGIN
  CASE mes OF
    1,3,5,7,8,10,12 : DataOK := (dia >= 1) AND (dia <= 31);
    4,6,9,11       : DataOK := (dia >= 1) AND (dia <= 30);
    2               : DataOK := (dia >= 1) AND
                          ((dia <= 28) OR ((dia = 29) AND ((any MOD 4) = 0)));
    ELSE DataOK := FALSE;
  END;
END;
```

```
(* Pausa : *)
(* Procediment que genera una pausa en el programa. *)
(* No es surt del procediment fins que no es premi la tecla ENTER. *)
PROCEDURE Pausa;
BEGIN
  WRITE('Premi <ENTER> per continuar');
  READLN;
END;
```

```
BEGIN
  WRITE('DIA : ');
  dia := RangDada(1,31);

  WRITE('MES : ');
  mes := RangDada(1,12);

  WRITE('ANY : ');
  any := RangDada(0,2000);

  IF DataOK(dia,mes,any) THEN
    WRITELN('Data correcta')
  ELSE
    WRITELN('Data incorrecta');

  Pausa;
END.
```

9.7 Donat el següent programa en Pascal:

```
PROGRAM A9_7;
USES vector;
VAR a : ARRAY [1..5] OF INTEGER;
```

```
    i : INTEGER;

BEGIN
  FOR i := 1 TO 5 DO a[i] := 2*i;
  WRITELN( sumatori(a) );
  WRITELN( mitjana(a) );
END.
```

Creeu la unitat `vector` que implementi els següents procediments i funcions:

- Un procediment que inicialitza a zero els elements d'un vector de qualsevol mida:

```
PROCEDURE Inicialitzar(VAR taula : ARRAY OF INTEGER);
```

- Una funció que retorna el resultat de calcular la suma de tots els elements d'un vector de qualsevol mida:

```
FUNCTION Sumatori(VAR taula : ARRAY OF INTEGER):LONGINT;
```

- Una funció que retorna el resultat de calcular la mitjana de tots els elements d'un vector de qualsevol mida:

```
FUNCTION Mitjana(VAR taula : ARRAY OF INTEGER):REAL;
```

Pascal disposa de dues funcions que permeten consultar quins són els límits inferior i superior d'un vector:

- `LOW(nom_vector)` ens retorna l'índex inferior del vector.
- `HIGH(nom_vector)` ens retorna l'índex superior del vector.

Programa en Pascal

```
UNIT vector;

INTERFACE

  (* Inicialitzar a zero els elements d'un vector de qualsevol mida *)
  PROCEDURE Inicialitzar(VAR taula : ARRAY OF INTEGER);

  (* Calcular la suma de tots els elements d'una taula *)
  FUNCTION Sumatori(VAR taula : ARRAY OF INTEGER):LONGINT;

  (* Calcular la mitjana de tots els elements d'una taula *)
  FUNCTION Mitjana(VAR taula : ARRAY OF INTEGER):REAL;

IMPLEMENTATION

  (* Als següents subprogrames no ens caldrà calcular LOW(taula) , *)
  (* ja que la base d'un array obert sempre és zero: LOW(taula) = 0 *)

  (* Inicialitzar a zero els elements d'un vector de qualsevol mida *)
  PROCEDURE Inicialitzar(VAR taula : ARRAY OF INTEGER);
  VAR i : INTEGER;
  BEGIN
    FOR i := 0 TO HIGH(taula) DO
      taula[i] := 0;
    END;
  END;

  (* Calcular la suma de tots els elements d'una taula *)
  FUNCTION Sumatori(VAR taula : ARRAY OF INTEGER):LONGINT;
  VAR i : INTEGER;
```

```

    total : LONGINT;
BEGIN
    total := 0;
    FOR i := 0 TO HIGH(taula) DO
        total := total + taula[i];
    Sumatori := total;
END;

(* Calcular la mitjana de tots els elements d'una taula *)
FUNCTION Mitjana(VAR taula : ARRAY OF INTEGER):REAL;
BEGIN
    Mitjana := Sumatori(Taula) / (HIGH(taula) + 1);
END;

END.
```

9.8 Sigui n un nombre enter de quatre xifres diferents. Definim les funcions:

- | $gran(n)$ com el nombre més gran que es pot formar amb les xifres de n .
- | $petit(n)$ com el nombre més petit que es pot formar amb les xifres de n .
- | $dif(n) = gran(n) - petit(n)$.

Per exemple, si tenim $n = 1984$, aleshores $gran(n) = 9841$, $petit(n) = 1489$ i $dif(n) = 8352$.

Dissenyeu un algorisme en pseudocodi i el corresponent programa en Pascal que, donat un enter n , comprovi que aquest té quatre xifres diferents i calculi i escrigui els valors de la següent successió fins que trobi un terme tal que $dif(k) = k$:

$$\begin{aligned}
 & dif(n) \\
 & dif(dif(n)) \\
 & dif(dif(dif(n)))
 \end{aligned}$$

Programa en Pascal

```

PROGRAM A9_8;

CONST
    NUM_XIFRES = 4;
    NUM_NOMBRES = 10000;

TYPE
    t_enter = LONGINT;
    t_vector = ARRAY [0..NUM_XIFRES-1] OF 0..9;

VAR
    nombre: t_enter;
    anterior: t_enter;

(* Converteix un enter en un vector que conté les seves xifres *)
PROCEDURE IntVect(n:t_enter; VAR v:t_vector);
VAR
    i: INTEGER;
```

```
BEGIN
  FOR i := 0 TO NUM_XIFRES-1 DO
    BEGIN
      v[i] := n MOD 10;
      n := n DIV 10;
    END;
END;

(* Converteix un vector que conté xifres en el corresponent nombre enter *)
PROCEDURE VectInt(VAR n:t_enter; v:t_vector);
VAR
  i: INTEGER;
BEGIN
  n := 0;
  FOR i := NUM_XIFRES-1 DOWNTO 0 DO
    n := n*10 + v[i];
  END;
END;

(* Endreça un vector en ordre creixent o decreixent *)
PROCEDURE Sort(VAR v:t_vector; creixent:BOOLEAN);
VAR
  i, j: INTEGER;
  aux: 0..9;
BEGIN
  FOR i := 0 TO NUM_XIFRES-2 DO
    FOR j := i+1 TO NUM_XIFRES-1 DO
      IF ((v[j] < v[i]) AND creixent) OR
          ((v[j] > v[i]) AND NOT(creixent)) THEN
        BEGIN
          aux := v[i];
          v[i] := v[j];
          v[j] := aux;
        END;
      END;
    END;
  END;
END;

(* Retorna el nombre més gran que es pot fer amb les xifres del paràmetre *)
FUNCTION gran(n:t_enter):t_enter;
VAR
  v: t_vector;
BEGIN
  IntVect(n, v);
  Sort(v, TRUE);
  VectInt(n, v);
  gran := n;
END;

(* Retorna el nombre més petit que es pot fer amb les xifres del paràmetre*)
FUNCTION petit(n:t_enter):t_enter;
VAR
  v: t_vector;
BEGIN
  IntVect(n, v);
  Sort(v, FALSE);
  VectInt(n, v);
  petit := n;
END;

(* Retorna la diferència entre el nombre més gran i més petit que es poden*)
```

```
(* escriure endreçant les xifres del nombre enter passat com a paràmetre *)
FUNCTION dif(n:t_enter):t_enter;
BEGIN
  dif := gran(n) - petit(n);
END;

(* Programa principal *)
BEGIN
  (* Llegim el nombre *)
  REPEAT
    WRITE('Introdueixi un nombre enter de ', NUM_XIFRES, ' xifres: ');
    READLN(nombre);
  UNTIL (nombre >= (NUM_NOMBRES DIV 10)) AND (nombre < NUM_NOMBRES);

  (* Iteracionem fins arribar a un valor fix *)
  REPEAT
    anterior := nombre;
    nombre := dif(nombre);
    Writeln('dif(',anterior, ') = ', nombre);
  UNTIL anterior = nombre;
END.
```

9.9 Imagineu un laberint format per una matriu quadrada de dimensions $n \times m$, amb k cel·les marcades que no es poden travessar. Dissenyau un algorisme en pseudocodi i el corresponent programa en Pascal que ens permeti trobar el camí més curt per viatjar de la cantonada (1,1) a la cantonada (n,m) sense passar per les cel·les marcades.

L'usuari introduirà els valors de n , de m i de k , així com les coordenades de les k cel·les marcades. La sortida serà les coordenades de les cel·les que formen el camí (tingueu en compte que pot no haver camí o haver més d'un).

Per exemple:

<u>Laberint</u>	<u>Entrada</u>	<u>Sortida</u>
	6 5 8	1 1
	1 2	2 1
	2 2	3 1
	2 4	4 1
	3 2	4 2
	4 4	4 3
	5 1	5 3
	5 2	6 3
	5 4	6 4
		6 5

Programa en Pascal

```
PROGRAM A9_9;

CONST
  MAX_FIL = 10; (* Màxim nombre de files de la matriu *)
```

```
MAX_COL = 10; (* Màxim nombre de columnes de la matriu *)
BARRERA = -1; (* Codi cel.la no travessable *)
INFINIT = MAX_FIL + MAX_COL; (* Codi inicialització cel.les *)

TYPE t_laberint = RECORD
  celles : ARRAY [ 1..MAX_FIL , 1..MAX_COL ] OF INTEGER;
  files : 1..MAX_FIL;
  columnes : 1..MAX_COL;
  barreres : 1..MAX_FIL*MAX_COL;
END;

TYPE t_coord_cella = RECORD
  f, c: INTEGER;
END;

TYPE t_veines = RECORD
  num_v : 0..8;
  coord : ARRAY [1..8] OF t_coord_cella;
END;

VAR
  i, j, k: INTEGER;
  L: t_laberint;

(* Procediment que troba les cel.les veïnes en connectivitat a 4 *)
PROCEDURE TrobaVeïnes(f, c: INTEGER; VAR veïnes: t_veïnes);
VAR i, j: INTEGER;
BEGIN
  (* Calculem les cel.les veïnes en connectivitat a quatre *)
  veïnes.num_v := 0;
  FOR i := f-1 TO f+1 DO
    FOR j := c-1 TO c+1 DO
      IF (i >= 1) AND (i <= L.files) AND (* dintre límits laberint *)
          (j >= 1) AND (j <= L.columnes) AND (* dintre límits laberint *)
          (L.celles[i,j] <> BARRERA) AND (* no cel.la barrera *)
          NOT ((i = f) AND (j = c)) AND (* no mateixa cel.la *)
          ((i = f) OR (j = c)) (* connectivitat a quatre *)
      THEN
        BEGIN
          veïnes.num_v := veïnes.num_v + 1;
          veïnes.coord[veïnes.num_v].f := i;
          veïnes.coord[veïnes.num_v].c := j;
        END;
      END;
    END;
  END;

(* Procediment que troba el camí mínim recursivament *)
PROCEDURE MiniDijkstra(f, c: INTEGER);
VAR
  i, j, k: INTEGER;
  veïnes: t_veïnes;
BEGIN
  (* Processem recursivament les cel.les veïnes *)
  TrobaVeïnes(f, c, veïnes);
  FOR k := 1 TO veïnes.num_v DO
    BEGIN
      i := veïnes.coord[k].f;
      j := veïnes.coord[k].c;
      IF (L.celles[i,j] > L.celles[f,c]+1) THEN
        BEGIN

```

```
        L.celles[i,j] := L.celles[f,c] + 1;
        MiniDijkstra(i,j);
    END;
END;

(* Procediment que escriu el camí mínim un cop trobat *)
PROCEDURE EscriuCami(f, c, f_inici, c_inici: INTEGER);
VAR
    i, j, k: INTEGER;
    veines: t_veines;
BEGIN
    IF (f = f_inici) AND (c = c_inici) THEN
        WRITELN('(',f,',',',',c,')')
    ELSE
        (* Cerquem entre les cel·les veïnes *)
        TrobaVeines(f, c, veines);
        FOR k := 1 TO veines.num_v DO
            BEGIN
                i := veines.coord[k].f;
                j := veines.coord[k].c;
                IF (L.celles[i,j] = L.celles[f,c]-1) THEN
                    BEGIN
                        EscriuCami(i, j, f_inici, c_inici);
                        WRITELN('(',f,',',',',c,')');
                    END;
                END;
            END;
        END;
    END;
END;

BEGIN
    WRITELN('Camí mínim dins un laberint');
    WRITELN('-----');

    (* Demanem el nombre de files i columnes del laberint *)
    WRITELN;
    REPEAT
        WRITE('Introdueixi les files i columnes del laberint : ');
        READLN( L.files , L.columnes );
    UNTIL (L.files >= 1) AND (L.files <= MAX_FIL) AND
        (L.columnes >= 1) AND (L.columnes <= MAX_COL);

    (* Inicialitzem les cel·les del laberint a un valor màxim *)
    FOR i := 1 TO L.files DO
        FOR j := 1 TO L.columnes DO
            L.celles[i,j] := INFINIT;
        END;
    END;

    (* Demanem les cel·les no travessables del laberint *)
    WRITELN;
    REPEAT
        WRITE('Introdueixi el nombre de barreres del laberint : ');
        READLN(L.barreres);
    UNTIL (L.barreres >= 1) AND (L.barreres <= L.files*L.columnes);

    FOR k := 1 TO L.barreres DO
        BEGIN
            REPEAT
                WRITE('Introdueix les coordenades (f,c) de la barrera ', k, ' : ');
                READLN(i , j);
            UNTIL (i >= 1) AND (i <= L.files) AND (j >= 1) AND (j <=
L.columnes);
            L.celles[i,j] := BARRERA;
        END;
    END;

    (* Partim de la cel·la (1,1) *)
```

```
L.celles[1,1] := 0;
MiniDijkstra(1, 1);

(* Imprimim el laberint amb els camins mínims *)
WRITELN;
WRITELN('El laberint és:');
FOR i := 1 TO L.files DO
  BEGIN
    WRITE([' ');
    FOR j := 1 TO L.columnes DO WRITE(L.celles[i,j]:3);
    WRITELN(' ');
  END;

(* Comprovem camí cap a la cel.la (#files,#columnes) *)
i := L.files;
j := L.columnes;
WRITELN;
IF L.celles[i,j] = INFINIT THEN
  WRITELN('El camí de (1,1) a (' ,i,',',j,') no existeix')
ELSE
  BEGIN
    WRITELN('El camí de (1,1) a (' ,i,',',j,') és:');
    EscriuCamí(i, j, 1, 1);
  END;
END.
```

Pràctica 10. Fitxers.

10.1 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que demani que introduïm noms d'alumne i la seva nota corresponent per teclat, i els guardi al fitxer 'alumnes.ftx'. Un cop haguem acabat d'introduir les dades, obri el fitxer en mode de lectura i imprimeixi per pantalla tots els noms i notes que conté.

Programa en PASCAL

```
PROGRAM A10_1;

TYPE t_alumne = RECORD
    nom : STRING[15];
    nota : REAL;
END;

VAR fitxer : FILE OF t_alumne;
    entrada : t_alumne;

BEGIN
    (* La variable fitxer treballarà amb el fitxer alumne.ftx *)
    ASSIGN(fitxer, 'alumnes.ftx');

    (* Creem el fitxer d'alumnes *)
    REWRITE(fitxer);

    (* Demana noms i notes i els grava al fitxer *)
    (* mentre no s'introdueix ENTER en el camp del nom *)
    WRITE('Introdueix un nom o ENTER per acabar : ');
    READLN(entrada.nom);
    WHILE entrada.nom <> '' DO
        BEGIN
            WRITE('Introdueix la nota corresponent a aquest alumne ');
            READLN(entrada.nota);
            WRITE(fitxer, entrada);          (* Escriu l'alumne al fitxer *)
            WRITE('Introdueix un nom o ENTER per acabar : ');
            READLN(entrada.nom);
        END;

    (* Tanca el fitxer *)
    CLOSE(fitxer);

    (* Obre el fitxer *)
    RESET(fitxer);

    (* Mentre no s'arribi al final del fitxer *)
    (* llegeix alumnes i els imprimeix *)
    WRITELN;
    WRITELN('Els alumnes que hi han al fitxer són:');
    WRITELN('-----');
    WHILE NOT EOF(fitxer) DO
        BEGIN
            READ(fitxer, entrada);          (* Llegeix un alumne del fitxer *)
            WRITELN(entrada.nom, ' ', entrada.nota:4:1);
        END;
    END;
```

```
(* Tanca el fitxer *)  
CLOSE(fitxer);  
  
READLN;  
END.
```

10.2 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que permetin, al fitxer d'alumnes creat a l'exercici 10.1, cercar un alumne pel seu camp nom i visualitzar per pantalla la seva nota.

Programa en PASCAL

```
PROGRAM A10_2;  
  
TYPE t_alumne = RECORD  
    nom : STRING[15];  
    nota : REAL;  
END;  
  
VAR fitxer      : FILE OF t_alumne;  
    entrada    : t_alumne;  
    nom_cercat : STRING[15];  
    trobat     : BOOLEAN;  
  
BEGIN  
    (* La variable fitxer treballarà amb el fitxer alumne.ftx *)  
    ASSIGN(fitxer, 'alumnes.ftx');  
  
    (* Demana el nom de l'alumne a buscar al fitxer *)  
    WRITE('Introdueix el nom de l'alumne a cercar : ');  
    READLN(nom_cercat);  
  
    (* Mentre no s'arribi al final del fitxer, llegeix *)  
    (* alumnes i compara el seu nom amb el nom cercat *)  
    trobat := FALSE;  
    RESET(fitxer);  
    WHILE (NOT EOF(fitxer)) AND (NOT trobat) DO  
        BEGIN  
            READ(fitxer, entrada);  
            trobat := (nom_cercat = entrada.nom);  
        END;  
  
    (* Comprova si l'ha trobat *)  
    IF trobat THEN  
        Writeln(entrada.nom, ' ', entrada.nota:4:1)  
    ELSE  
        Writeln('Aquest alumne no es troba dins el fitxer');  
  
    (* Tanca el fitxer *)  
    CLOSE(fitxer);  
  
    READLN;  
END.
```

10.3 Dissenyeu un algorisme en pseudocodi i realitzeu el corresponent programa en Pascal que permetin, al fitxer d'alumnes creat a l'exercici 10.1, cercar un alumne per la seva posició dins el fitxer i visualitzar per pantalla les seves dades.

Suposarem que el primer alumne del fitxer és el número 1.

Feu servir el procediment `SEEK(fitxer, posició)` i la funció `FILESIZE(fitxer)` de Turbo Pascal.

Programa en PASCAL

```
PROGRAM A10_3;

TYPE t_alumne = RECORD
    nom : STRING[15];
    nota : REAL;
END;

VAR fitxer      : FILE OF t_alumne;
    entrada     : t_alumne;
    num_alumne  : LONGINT;

BEGIN
    (* La variable fitxer treballarà amb el fitxer alumne.ftx *)
    ASSIGN(fitxer, 'alumnes.ftx');
    RESET(fitxer);

    (* Demana el número d'alumne a recuperar del fitxer. *)
    (* Suposem que el primer alumne del fitxer és el 1. *)
    REPEAT
        WRITE('Introdueix el número de l'alumne a llistar : ');
        READLN(num_alumne);
        IF (num_alumne < 1) OR (num_alumne > FILESIZE(fitxer)) THEN
            WRITELN('Número d'alumne incorrecte.');
```

```
        UNTIL (num_alumne >= 1) AND (num_alumne <= FILESIZE(fitxer));

        (* Es posiciona al fitxer i llegeix l'alumne. *)
        SEEK(fitxer, num_alumne-1);
        READ(fitxer, entrada);
        (* També es pot fer, encara que de manera més ineficient, *)
        (* llegint tots els registres fins arribar al que volem *)
        (* (suposem que tinguem declarada la variable 'i') : *)
        (* FOR i := 1 TO num_alumne DO READ(fitxer, entrada); *)

        (* Escriu l'alumne per pantalla *)
        WRITELN(entrada.nom, ' ', entrada.nota:4:1);

        (* Tanca el fitxer *)
        CLOSE(fitxer);

        READLN;
    END.
```

10.4 Amplieu l'exercici 9.4 per tal que permeti crear un fitxer que contindrà noms i coordenades de punts de l'espai bidimensional, i que permetrà posteriorment calcular el centre de gravetat d'aquests punts.

Un punt estarà format per un nom que l'identifica i les seves coordenades:

t_punt

id	x	y
----	---	---

En iniciar el programa les opcions que ens apareixeran seran:

1. Afegir punts a un fitxer. 2. Llistar un fitxer de punts. 3. Modificar un punt d'un fitxer. 4. Calcular el centre de gravetat. 5. Finalitzar.

- | La primera opció permetrà obrir un fitxer de punts (o crear un fitxer nou de punts, en cas que el fitxer que ens indiqui l'usuari no existeixi) i introduir nous punts.
- | La segona opció ens permetrà obrir un fitxer de punts existent i llistar tots els punts que conté.
- | La tercera opció ens permetrà obrir un fitxer de punts existent i realitzar una cerca d'un determinat punt pel camp de l'identificador. Un cop trobat, podem modificar les seves coordenades i tornar a escriure-ho al fitxer.
- | La quarta opció ens permetrà obrir un fitxer de punts existent i realitzar el càlcul del centre de gravetat.
- | La cinquena opció és per sortir del programa.
- | I podeu afegir tot de noves opcions al menú que penseu convenientes: càlcul de la recta de regressió dels punts, càlcul de les cotes dels punts, càlcul del cercle de radi mínim que conté els punts, ordenar els punts pel camp identificador, dibuixar els punts per pantalla, etc.

El disseny de l'algorisme ha de ser modular. Feu servir funcions i subrutines per implementar les diferents opcions del menú.

Programa en PASCAL

```
PROGRAM A10_4;

USES crt;

TYPE
  t_id_punt = STRING[15];
  t_punt = RECORD
    id    : t_id_punt;
    x , y : REAL;
  END;

VAR
  nom_fitxer  : STRING;
  centre_grav : t_punt;
  opcio       : 1..5;

(* ExisteixFitxer : *)
(* Funció booleana que retorna cert si existeix el fitxer i *)
(* fals en cas contrari. Tanca el fitxer si aquest existeix. *)
FUNCTION ExisteixFitxer(NomFitxer:STRING): BOOLEAN;
VAR Fitxer : FILE;
```

```

BEGIN
  {$I-}
  ASSIGN(Fitxer, NomFitxer);
  RESET(Fitxer);
  CLOSE(Fitxer);
  {$I+}
  ExisteixFitxer := (IOResult = 0) AND (NomFitxer <> '');
END;

(* IntroduirPunts : *)
(* Procediment que demana punts per pantalla i els escriu al fitxer *)
(* passat com a paràmetre. Si aquest no existeix, el crea. *)
PROCEDURE IntroduirPunts(NomFitxer:STRING);
VAR fitxer : FILE OF t_punt;
    punt : t_punt;
BEGIN
  (* Si existeix el fitxer, l'obre i es posiciona al final per *)
  (* afegir nous punts. Si no existeix el fitxer, el crea. *)
  IF ExisteixFitxer(nom_fitxer) THEN
    BEGIN
      ASSIGN(fitxer, nom_fitxer);
      RESET(fitxer);
      SEEK(fitxer, FILESIZE(fitxer));
    END
  ELSE
    BEGIN
      ASSIGN(fitxer, nom_fitxer);
      REWRITE(fitxer);
    END;

  (* Demana noms i coordenades i els grava al fitxer *)
  (* mentre no s'introdueix ENTER en el camp del nom *)
  WRITE('Introdueix el nom del punt o ENTER per acabar : ');
  READLN(punt.id);
  WHILE punt.id <> ' ' DO
    BEGIN
      WRITE('Introdueix les coordenades corresponents a aquest punt:');
      READLN(punt.x, punt.y);
      WRITE(fitxer, punt);
      WRITE('Introdueix el nom del punt o ENTER per acabar : ');
      READLN(punt.id);
    END;
  CLOSE(fitxer);
END;

(* LListarPunts : *)
(* Procediment que llista per pantalla tots els punts *)
(* del fitxer passat com a paràmetre. *)
PROCEDURE LlistarPunts(NomFitxer:STRING);
VAR fitxer : FILE OF t_punt;
    punt : t_punt;
BEGIN
  (* Si el fitxer existeix, llegeix els punts un a un, *)
  (* escrivint-los per pantalla. *)
  IF ExisteixFitxer(nom_fitxer) THEN
    BEGIN
      ASSIGN(fitxer, nom_fitxer);
      RESET(fitxer);
      WHILE NOT EOF(fitxer) DO
        BEGIN
          READ(fitxer, punt);
          WRITELN(punt.id, ' : (', punt.x:4:2, ', ', punt.y:4:2, ')');
        END;
      CLOSE(fitxer);
    END
  END

```

```

ELSE
  Writeln('Aquest fitxer de punts no existeix.');
```

```

END;
```

```

(* ModificarPunt : *)
(* Procediment que demana l'identificador d'un punt i el busca al *)
(* fitxer passat com a paràmetre. Quan el troba, demana les noves *)
(* coordenades i el torna a gravar en el fitxer. *)
PROCEDURE ModificarPunt(NomFitxer:STRING);
VAR fitxer : FILE OF t_punt;
    punt : t_punt;
    id_punt : t_id_punt;
BEGIN
  (* Si el fitxer existeix, llegeix els punts *)
  (* un a un fins trobar el punt buscat. *)
  IF ExisteixFitxer(nom_fitxer) THEN
    BEGIN
      ASSIGN(fitxer, nom_fitxer);
      RESET(fitxer);
      WRITE('Introdueixi l'identificador del punt a buscar : ');
      READLN(id_punt);
      punt.id := '';
      WHILE (NOT EOF(fitxer)) AND (id_punt <> punt.id) DO
        READ(fitxer, punt);
        IF id_punt = punt.id THEN
          BEGIN
            (* Si l'ha trobat, l'escriu per pantalla, demana les *)
            (* noves coordenades i escriu el nou punt a la mateixa *)
            (* posició del fitxer. *)
            Writeln(punt.id, ' : (', punt.x:4:2, ', ', punt.y:4:2, ')');
            WRITE('Introdueix les noves coordenades d'aquest punt : ');
            READLN(punt.x, punt.y);
            SEEK(fitxer, FILEPOS(fitxer)-1);
            WRITE(fitxer, punt);
            END
          ELSE
            Writeln('Punt no trobat dins el fitxer.');
```

```

          CLOSE(fitxer);
          END
        ELSE
          Writeln('Aquest fitxer de punts no existeix.');
```

```

      END;

(* CalcularCentreGrav : *)
(* Funció que calcula el centre de gravetat dels punts del fitxer *)
(* passat com a paràmetre. Si pot calcular el centre de gravetat, *)
(* el retorna a la variable 'centre_grav' i la funció retorna TRUE. *)
(* Si no pot calcular el centre de gravetat la funció retorna FALSE. *)
FUNCTION CalcularCentreGrav(NomFitxer:STRING; VAR
centre_grav:t_punt):BOOLEAN;
VAR fitxer : FILE OF t_punt;
    punt : t_punt;
    num_punts : INTEGER;
BEGIN
  (* Si el fitxer existeix, llegeix els punts un a un, *)
  (* calculant el centre de gravetat. *)
  IF ExisteixFitxer(nom_fitxer) THEN
    BEGIN
      ASSIGN(fitxer, nom_fitxer);
      RESET(fitxer);
      centre_grav.id := 'Centre grav';
      centre_grav.x := 0;
      centre_grav.y := 0;
      num_punts := 0;
      WHILE NOT EOF(fitxer) DO
```

```
        BEGIN
        READ(fitxer, punt);
        centre_grav.x := centre_grav.x + punt.x;
        centre_grav.y := centre_grav.y + punt.y;
        num_punts := num_punts + 1;
        END;
    (* Comprovem que almenys hi hagués un punt *)
    IF num_punts > 0 THEN
        BEGIN
            centre_grav.x := centre_grav.x / num_punts;
            centre_grav.y := centre_grav.y / num_punts;
            CalcularCentreGrav := TRUE;
        END
    ELSE
        CalcularCentreGrav := FALSE;
    CLOSE(fitxer);
    END
ELSE
    CalcularCentreGrav := FALSE;
END;

BEGIN (* Programa principal *)

    REPEAT
        CLRSCR;
        WRITELN('Menú');
        WRITELN('-----');
        WRITELN('1. Introduir punts a un fitxer  ');
        WRITELN('2. Llistar punts d'un fitxer  ');
        WRITELN('3. Modificar un punt d'un fitxer');
        WRITELN('4. Calcular el centre de gravetat');
        WRITELN('5. Finalitzar  ');
        WRITELN('-----');
        WRITE('Opció : ');
        READLN(opcio);

        CASE opcio OF
            1 : BEGIN
                WRITELN;
                WRITE('Introdueixi el nom del fitxer de punts : ');
                READLN(nom_fitxer);
                IntroduirPunts(nom_fitxer);
                END;

            2 : BEGIN
                WRITELN;
                WRITE('Introdueixi el nom del fitxer de punts : ');
                READLN(nom_fitxer);
                LlistarPunts(nom_fitxer);
                END;

            3 : BEGIN
                WRITELN;
                WRITE('Introdueixi el nom del fitxer de punts : ');
                READLN(nom_fitxer);
                ModificarPunt(nom_fitxer);
                END;

            4 : BEGIN
                WRITELN;
                WRITE('Introdueixi el nom del fitxer de punts : ');
                READLN(nom_fitxer);
                IF CalcularCentreGrav(nom_fitxer, centre_grav) THEN
                    WRITELN('El centre de gravetat és (',
                        centre_grav.x:4:2, ',', centre_grav.y:4:2, ')')
                ELSE

```

```
        WRITELN('Error: fitxer buit o no existeix el fitxer');
    END;

    5 : BEGIN
        WRITELN;
        WRITELN('Gràcies per utilitzar CGRAV v 1.0 !');
        END;

    ELSE
        WRITELN('Opció incorrecta');

    END;

    WRITE('Premi <ENTER> per continuar ');
    READLN;

    UNTIL opcio = 5;

END.
```

Pràctica 11. Exercicis finals.

11.1 Volem tenir una base de dades amb tots els alumnes de l'assignatura Fonaments d'Informàtica d'un trimestre, on constaran els noms, grup, notes, etc. Aquesta informació es guardarà a un fitxer i en tot moment la podrem consultar o modificar. Es tracta de realitzar un algorisme en pseudocodi i el corresponent programa en Pascal que ens permetin gestionar aquestes dades.

Les fitxes dels alumnes contindran la següent informació:

```
CONST MAX_NOTES = 5;  
  
TYPE t_alumne = RECORD  
    Especialitat: STRING[15];  
    Grup: STRING[4];  
    Nom: STRING[30];  
    Notes: ARRAY [1..MAX_NOTES] OF REAL;  
END;
```

El programa o algorisme mostrarà un menú amb les següents opcions:

- | |
|--|
| <ol style="list-style-type: none">1. Obrir un fitxer d'alumnes.2. Altes de nous alumnes.3. Baixes d'alumnes.4. Modificacions de les dades d'un alumne.5. Llistat dels alumnes del fitxer.6. Ordenació dels alumnes pel camp nom.0. Finalitzar. |
|--|

i el programa no acabarà fins que l'opció escollida per l'usuari sigui finalitzar.

Podrem tenir diferents fitxers d'alumnes. Això ens permetrà guardar en fitxers diferents les dades dels alumnes de cada trimestre i any. L'usuari escollirà amb la primera opció del menú el nom del fitxer amb el que vol treballar. Si ja s'estava treballant amb un fitxer, tancarem aquest i obrirem el nou fitxer escollit.

El disseny del programa i algorisme haurà de ser modular. Serà necessari definir els subprogrames que realitzin les següents accions:

- | Obrir un fitxer, donat el seu nom passat com a paràmetre. Si no existeix el fitxer, crear-lo.
- | Visualitzar per pantalla les dades d'un alumne passat com a paràmetre, incloent la seva mitjana de notes, que no hi és dins el fitxer (s'haurà de calcular).
- | Cercar un alumne dins el fitxer, donat el seu nom passat com a paràmetre, i retornar la seva posició. Si no ha estat trobat, retornar -1.

- | Donar d'alta nous alumnes, escrivint-los dins el fitxer. Si ha estat possible, retornar el valor booleà `TRUE`, si no ha estat possible (l'alumne ja existia o error d'entrada/sortida al fitxer) retornar el valor booleà `FALSE`.
- | Donar de baixa alumnes existents, esborrant-los del fitxer i reorganitzant el fitxer de nou. Si ha estat possible, retornar el valor booleà `TRUE`, si no ha estat possible (l'alumne no existia o error d'entrada/sortida al fitxer) retornar el valor booleà `FALSE`.
- | Modificar les dades dels alumnes existents, escrivint les noves dades al fitxer. Si ha estat possible, retornar el valor booleà `TRUE`, si no ha estat possible (l'alumne no existia o error d'entrada/sortida al fitxer) retornar el valor booleà `FALSE`.
- | Llistat dels alumnes del fitxer, utilitzant el subprograma anterior per visualitzar un alumne per pantalla.
- | Ordenació dels alumnes del fitxer pel camp nom.

11.2 Simulacions mitjançant un autòmat cel·lular: "el joc de la vida".

Un laboratori d'investigació distribueix una colònia de bacteris en un cultiu, que es pot considerar una superfície quadriculada de N files i M columnes. Cada casella d'aquesta superfície pot estar buida o contenir un bacteri. A partir d'una configuració inicial de bacteris en caselles, la colònia evoluciona generació a generació segons unes lleis genètiques determinades que depenen del nombre de veïns que tingui cada casella:

- Naixement: a cada casella buida que tingui exactament tres caselles veïnes ocupades per bacteris hi naixerà un nou bacteri.
- Mort per aïllament: tot bacteri que ocupi una casella amb cap o un veí mor per aïllament.
- Mort per asfíxia: tot bacteri que ocupi una casella amb més de tres veïns mor per asfíxia.
- Només sobreviuen els bacteris que tinguin dos o tres veïns.

La casella que ocupa la fila i i la columna j s'identifica mitjançant (i, j) , on $1 \leq i \leq N$, $1 \leq j \leq M$. Els veïns d'una casella (i, j) són les posicions adjacents $(i-1, j-1)$, $(i-1, j)$, $(i-1, j+1)$, $(i, j-1)$, $(i, j+1)$, $(i+1, j-1)$, $(i+1, j)$ i $(i+1, j+1)$ que estan compreses dins la superfície i que estan ocupades per un bacteri.

Els bacteris que neixen o moren no afecten fins que s'ha completat un cicle evolutiu, entenent per aquest un cicle en el que s'ha decidit la supervivència o mort dels insectes (vius en començar el cicle) d'acord a les lleis genètiques mencionades.

Així, a una superfície 4x4, la colònia de l'esquerra de la següent figura evoluciona a les dues pròximes generacions tal i com es mostra:

```

. . * .      . * * .      . * . *      . . . .
. * * .      . * * *      . . . .      . . . .
. . * .      . . * *      . * . *      . . . .
. . * .      . . . .      . . . .      . . . .

```

Es demana simular l'evolució d'una colònia inicial de bacteris durant un cert nombre de transicions. Els passos a seguir seran els següents:

1. Demanar la configuració del tauler. Aquesta es pot introduir per teclat o residir en un fitxer. La configuració té el següent format:
 - Els dos primers elements són dos nombres enters que indiquen el nombre de files i columnes del tauler. Cal comprovar que aquest dos valors no excedeixin les dimensions màximes del tauler.
 - Els següents elements seran parelles de nombres que indicaran on hi ha una posició del tauler ocupada. Cal comprovar que els seus valors no surtin fora del tauler definit pels dos valors anteriors.
2. Simular generacions i visualitzar-les per pantalla fins que l'usuari premi una tecla especial, per exemple ESC, per finalitzar. Per exemple

<pre> EL JOC DE LA VIDA Premi F per llegir una configuració d'un fitxer. Premi T per introduir una configuració per teclat. T 5 5 2 2 3 2 3 3 3 4 </pre>	<pre> Generació 0 * * * * Premi: F per guardar en fitxer C per la següent generació ESC per finalitzar </pre>	<pre> Generació 1 * * * * Premi: F per guardar en fitxer C per la següent generació ESC per finalitzar </pre>
--	---	---

La colònia de bacteris serà una matriu d'elements del tipus cel·la. Les seves dimensions màximes venen definides per les constants MAXFIL i MAXCOL que definim a l'inici de l'algorisme.

El tipus cel·la serà un registre format per una variable que indicarà si està ocupada o buida i un altre variable que guardarà el nombre de veïns que té. D'aquesta manera, per calcular la següent generació podrem recórrer la matriu cel·la a cel·la i comptar el nombre de veïns que té. A continuació tornem a recórrer la matriu marcant com a buides les cel·les dels elements que moren i marcant com a ocupades les cel·les dels nous elements que neixen.

És aconsellable (i obligatori) la utilització de subrutines en la realització de la pràctica. Per exemple:

- Una funció que retorni un nombre enter llegit pel teclat comprovant que es troba entre dos valors passats com a paràmetres de la funció. Es farà servir per llegir les dimensions de la colònia i les caselles ocupades.
- Un procediment que llegeixi la colònia d'un fitxer i un altre que la guardi.
- Un procediment que visualitzi la colònia per pantalla.
- Una funció que retorni el nombre de veïns d'una cel·la.
- Un procediment que calculi la següent generació.
- I totes les funcions i procediments que es creguin convenients per afavorir la simplicitat i llegibilitat del programa i la reutilització del codi.