TEXTOS DOCENTS

328

INTRODUCCIÓ A LA PROGRAMACIÓ EN FORTRAN 77

Juan Carlos Paniagua
Albert Solé

Departament de Química Física

UNIVERSITAT DE BARCELONA

TEXTOS DOCENTS

328

INTRODUCCIÓ A LA PROGRAMACIÓ EN FORTRAN 77

Juan Carlos Paniagua
Albert Solé

Departament de Química Física

Publicacions i Edicions

UNIVERSITAT DE BARCELONA

Adolf Florensa, s/n, 08028 Barcelona; tel. 934 035 430; fax 934 035 531;
comercial.edicions@ub.edu; www.publicacions.ub.edu

ISBN: 978-84-9168-027-7

Aquesta publicació ha rebut l'ajut de la Generalitat de Catalunya.

PRÒLEG

Aquest llibre pretén proporcionar al lector unes nocions d'informàtica molt bàsiques i introduir-lo en un llenguatge de programació d'àmbit cientificotècnic. El text està orientat a no professionals de la informàtica —estudiants, graduats i llicenciats en carreres científiques o tècniques (física, química, biologia, geologia, enginyeries, etc.)— que vulguin tenir unes nocions de programació suficients per poder elaborar programes senzills o de dificultat mitjana. Se suposarà que l'entorn de treball és el que proporcionen les distribucions del sistema operatiu GNU/Linux, el qual pot estar instal·lat al disc dur o bé carregar-se des d'un CD autònom (*live CD*). El llenguatge escollit és el Fortran 77, que és clar i senzill, i està molt estès en el camp de les ciències fisicoquímiques. El fet que s'hagi d'escollir un llenguatge concret per poder posar en pràctica el material desenvolupat és poc rellevant, perquè tots els llenguatges de programació d'alt nivell són essencialment equivalents en el nivell introductori d'aquest text.

En cap moment hem pretès presentar una exposició completa del llenguatge Fortran; per a això hi ha els manuals dels compiladors corresponents. El lector amb certa experiència en programació detectarà, sens dubte, certes mancances en el contingut (descripcions incompletes, excepcions no indicades, instruccions no esmentades, etc.), que es justifiquen pel nostre afany d'assolir un redactat senzill i pedagògic. Per exemple, malgrat que la instrucció *open* admet un nombre considerable de paràmetres, només introduirem els dos que són indispensables. D'altra banda, l'ordre de l'exposició s'ha fixat d'acord amb criteris més pedagògics que lògics. Així, el *do* implícit s'introdueix en relació amb les variables indexades —que és on té les seves principals aplicacions— en lloc de fer-ho quan s'expliquen les instruccions d'entrada i sortida —que és on li correspondria des d'un punt de vista lògic.

Els conceptes es presenten de manera esglaonada: s'introdueixen amb l'ajut d'exemples senzills perquè el lector capti de seguida la seva utilitat i, a mesura que avança l'explicació, es van tractant aplicacions més complexes. Cap al final del llibre s'inclouen exercicis amb un grau de dificultat major, que combinen instruccions i algorismes desenvolupats separadament en les seccions anteriors. La major part dels exercicis proposats inclouen la solució o bé un conjunt de dades i resultats que permetin verificar el funcionament correcte del programa. En general, els resultats numèrics s'indiquen amb sis xifres significatives, independentment de la precisió de les dades i de la que proporcioni realment el programa. Això fa que el lector pugui trobar petites diferències en els seus resultats, depenent de la instal·lació (processador, distribució de GNU/Linux i versió del compilador) i de la precisió de les variables que utilitzi.

El nostre objectiu és que el profà en la matèria pugui iniciar-se de seguida en la pràctica de la programació i aconsegueixi un nivell que li permeti abordar sense dificultats la lectura de manuals més complets. Volem fer que la programació resulti fàcil per al que s'hi enfronta per primera vegada, de manera que li perdi la por i pugui apreciar les possibilitats que ofereix aquesta eina informàtica. Si, a més d'això, aconseguim engrescar el lector perquè aprofundeixi en el llenguatge Fortran o en qualsevol altre, creiem que estarà en disposició de fer-ho de manera autodidacta o bé assistint a cursos avançats de programació.

Aquesta obra té un clar precedent en el llibre *Introducció a la programació per a químics*, la primera edició del qual fou publicada el 1994 per Publicacions de la Universitat de Barcelona i sub-

vencionada mitjançant un ajut per a projectes d'innovació universitària concedit pel Vicerectorat d'Ensenyament i Professorat de la Universitat de Barcelona a proposta del Gabinet d'Avaluació i Innovació Universitària. D'aquest llibre, se n'han publicat quatre edicions, l'última el 1997. L'experiència acumulada durant els anys d'utilització en les assignatures "Aplicació de la Informàtica a Problemes Químics" i "Càlcul Numèric i Eines Informàtiques" de la llicenciatura en Química, i "Aplicació de la Informàtica a Problemes d'Enginyeria Química" i "Informàtica Aplicada" de l'ensenyament d'Enginyeria Química, juntament amb el canvi que ha suposat la creixent popularització del programari lliure en aquest període, ens han portat a confeccionar el present llibre a partir d'aquella base. El nou text està completament reescrit i reestructurat; algunes seccions d'escassa utilitat han estat eliminades i se n'han afegit de noves, tot buscant millorar les qualitats pedagògiques de l'exposició i cobrir un ventall d'interessos més ampli.

La llista de persones que han contribuït directament o indirectament en aquest text és llarga, ja que inclou tots els que van col·laborar d'una manera o altra en el llibre d'*Introducció a la programació per a químics* i altres que han participat en les assignatures esmentades des que s'inicià la implantació del Pla d'estudis de 1992 i han fet suggeriments diversos. Hem de destacar, no obstant això, Pere Alemany, Josep Maria Lucas, Eudald Vilaseca, Fernando Mota, Francesc Mas, Margarita Albertí, Ramón Sayós, Antonio Aguilar, Jaime Rubio, Juan José Novoa, Francesc Illas, Xavier Giménez, Àngels Povill, Carme Sousa i Laura López-Tomàs. També volem reconèixer la paciència amb què l'Eudald ha suportat les interminables sessions de revisió de manuscrits en el despatx que comparteix amb un dels autors (JCP).

Juan Carlos Paniagua i Albert Solé
Departament de Química Física de la Universitat de Barcelona
Barcelona, 2007

Índex

1 Nocions d'informàtica

En aquest capítol s'introduiran els termes més emprats en l'àmbit informàtic amb un llenguatge molt planer, la qual cosa ens farà sacrificar, de vegades, el rigor en favor de la simplicitat. Aquesta introducció serà força incompleta, atesos els objectius d'aquest text i l'amplitud del camp; a més es tracta d'una matèria que evoluciona molt ràpidament, per la qual cosa remetem el lector interessat a ampliar o actualitzar la informació proporcionada a les referències recollides a la bibliografia. Com que les obres impreses necessiten més temps per posar-se al dia que les publicades a través de la xarxa Internet, hem optat per incloure més referències del segon tipus. D'altra banda, una opció pràctica per consultar el significat d'un terme determinat és fer una cerca estesa a tota la xarxa mitjançant un buscador de paraules (*www.google.com*, *search.yahoo.com*, *www.live.com*, etc.).

1.1 L'ordinador

Un *ordinador* és una màquina capaç d'entendre instruccions externes per realitzar una tasca, fer-la i mostrar-ne el resultat. La informació subministrada pot consistir en dades numèriques, paraules (teclejades o seleccionades dins d'un menú o d'una finestra), imatges, so, etc., i la tasca pot implicar operacions algebraiques, comparacions entre dades, còpia d'informació, etc. De fet, hi ha molts aparells capaços d'entendre certes instruccions per realitzar determinades tasques (un equip de música, una rentadora, etc.), però el que caracteritza els ordinadors és la seva versatilitat; és a dir, la varietat d'instruccions i tasques que admeten. La *informàtica* estudia el disseny d'ordinadors i les seves aplicacions.

En qualsevol ordinador podem distingir dos tipus de components: el maquinari i el programari. El *maquinari* o *hardware* és el conjunt de components físics de l'ordinador; és a dir, tot allò que es pot tocar: teclat, monitor, circuits electrònics,[1] etc. Anomenem *programari* o *software* les instruccions (components lògics) que permeten manipular el maquinari en la forma desitjada (llegir el contingut d'un disc magnètic o enregistrar-hi informació, efectuar operacions matemàtiques amb dades introduïdes a través del teclat, imprimir-ne el resultat en una impressora, retocar una imatge, etc.). De vegades s'inclou també sota la denominació *software* la informació emmagatzemada a l'ordinador (textos, dades numèriques, imatges, so, etc.).

Per clarificar els conceptes de maquinari i programari podem emprar el símil d'un receptari de cuina: les receptes (intangibles) contingudes en el llibre equivaldrien al programari, i el paper i la tinta (tangibles) constituirien el maquinari.

1.2 El maquinari

Els components de maquinari que conté un ordinador poden ser molt diversos, però sempre hi ha un processador, una memòria ROM, una memòria RAM i uns perifèrics (figura 1.1).

[1] Compte amb les enrampades: certs components interns dels ordinadors poden produir descàrregues elèctriques fins i tot amb l'aparell desendollat.

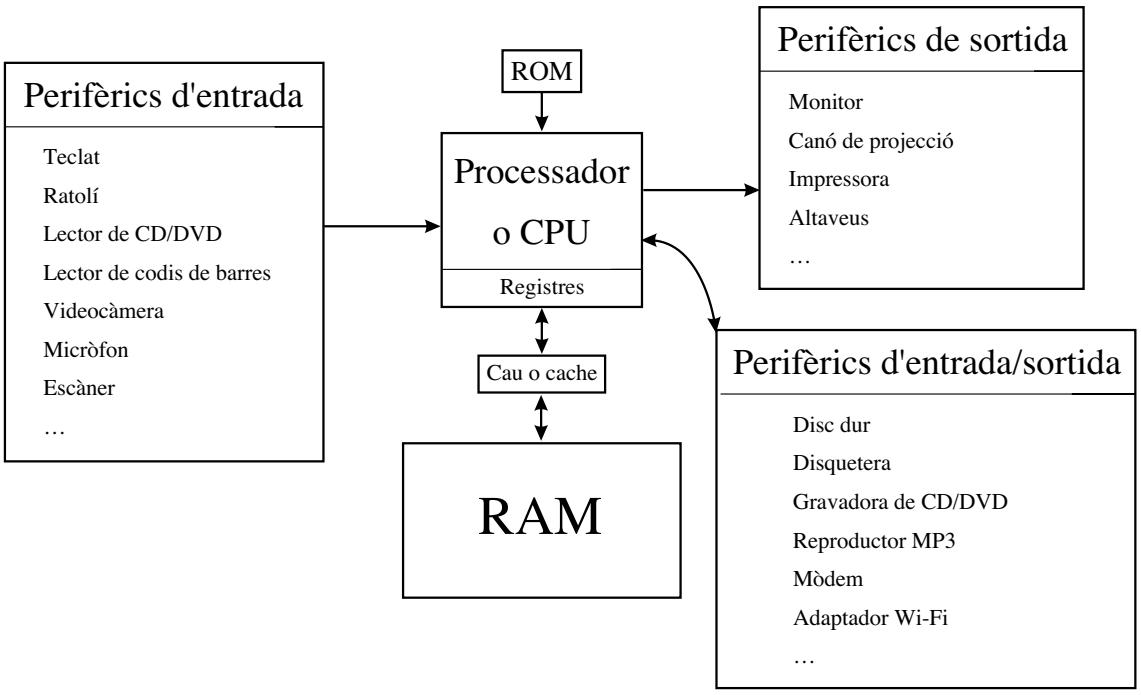

Figura 1.1: Esquema del maquinari d'un ordinador.

El *processador* o *CPU* (Central Processing Unit: unitat central de procés) és l'òrgan director del funcionament de l'ordinador (el "cervell" de la màquina). En endavant utilitzarem preferentment la paraula *processador*, atès que, en el llenguatge del carrer, se sol anomenar CPU o *torre* la capsa que conté la major part dels components d'un ordinador no portàtil (exclosos el teclat, el ratolí, el monitor, la impressora, etc.). Els processadors que porten actualment la majoria d'ordinadors —tant els d'ús domèstic o *personals* (PC: Personal Computer) com els d'ús professional— són *escalars*, en el sentit que fan les operacions d'una en una (Intel Core, AMD Athlon...). Alguns ordinadors, però, tenen processadors *vectorials*, els quals poden fer una mateixa operació sobre diferents dades alhora, processadors amb més d'un *core* (Dual Core, Quad Core...), que funcionen com si fossin conjunts de processadors, o diversos processadors que treballen de manera coordinada (*SMP*: Symmetric Multi-Processing). El ritme de treball d'un processador ve determinat per un rellotge amb una freqüència fixa que, actualment, se sol mesurar en gigahertzs (1 GHz = 10^9 Hz). Per a una mateixa família de processadors —per exemple, els Pentium 4— la velocitat de procés augmenta amb la freqüència del rellotge, però aquesta dada no permet comparar rendiments de famílies de processadors diferents —per exemple, un Pentium 4 amb un Pentium M o amb un Athlon. Aquesta comparació s'ha de fer a partir dels temps que triguen a fer una mateixa tasca. En l'àmbit científic s'utilitza molt el nombre de *flops* per comparar el rendiment dels processadors. Un *flops* (Floating-Point Operations per Second) és una operació amb nombres reals —també anomenats de *coma flotant*— per segon i, com que els processadors moderns són molt ràpids, s'utilitzen múltiples d'aquella quantitat: megaflops (1 Mflops = 10^6 flops), gigaflops (1 Gflops = 10^9 flops), etc.

La informació que arriba al processador i la que aquest produeix es guarda en els *registres*, que són petites zones de memòria temporal d'accés molt ràpid. Així, si el processador ha de sumar dos nombres, cada un d'aquests es guardarà en un registre i, una vegada calculada la seva suma, el resultat queda guardat en un altre registre (que pot ser el mateix que guardava un dels sumands). Com que la capacitat dels registres és molt petita, el processador necessita poder accedir a zones de memòria més grans, com ara la memòria RAM. La comunicació entre el processador i altres parts de l'ordinador es fa mitjançant un *bus*, que és un conjunt de conduccions elèctriques (16, 32, 64..., depenent de la màquina) que transmeten informació de manera simultània (*en paral·lel*).

Anomenem *memòria ROM* (Read Only Memory) una zona de memòria que no es pot modificar i que no s'esborra en apagar l'ordinador. Aquesta memòria conté instruccions molt bàsiques per al

funcionament de l'ordinador. Així, quan l'engeguem, el processador llegeix les instruccions emmagatzemades en la ROM, les quals poden fer que aquell busqui instruccions per acabar d'arrencar en algun disquet o CD que haguem introduït prèviament o en el disc dur, etc.

Anomenem *memòria RAM* (Random Access Memory)[2] o *principal* una zona de memòria que es pot modificar i que s'esborra quan apaguem l'ordinador. Es diu, per això, que la informació que guarda aquesta memòria és *volàtil*. A la memòria RAM —que és molt més gran que la ROM— s'emmagatzemen les instruccions que indiquen al processador les tasques que volem que faci, així com les dades necessàries per realitzar-les i els resultats que puguin produir aquestes tasques.

En la major part de les calculadores, les memòries de treball són de tipus RAM (podem guardar-hi dades, esborrar-les, etc.); en canvi, les instruccions per calcular funcions trigonomètriques, logaritmes, etc. estan enregistrades en memòria ROM, ja que no s'esborren en apagar la calculadora. La informació impresa en paper proporciona un símil no electrònic per a aquests dos tipus de memòria: un llibre de la biblioteca es pot considerar un suport de memòria ROM, ja que la informació que conté pot ser llegida però no podem alterar-la; una llibreta, en canvi, representaria una memòria RAM, en la qual podem anotar dades, esborrar-les, etc. (tot i que no té el caràcter volàtil de la RAM de l'ordinador).

Quan comprem un ordinador no solament hem de fixar-nos en la capacitat de la seva memòria RAM; és important que aquesta es pugui ampliar afegint *mòduls* o *xips*[3] de memòria o substituint els que té per uns altres de més capacitat, ja que les noves versions del programari suposen, sovint, un augment en les exigències de memòria.

El ritme de treball del processador és més ràpid que l'accés a la memòria RAM, per la qual cosa se sol intercalar entre aquells dos components una petita memòria d'accés molt ràpid (i més cara) que s'anomena *cau* o *cache*. Aquesta acumula les dades transferides al processador o des del processador en blocs per tal d'augmentar la velocitat de transferència.

Els *perifèrics* o *dispositius* permeten que el processador es comuniqui amb l'exterior. N'hi ha de molts tipus, però es poden classificar en tres grups:

- els *d'entrada* permeten transferir informació al processador: el teclat, el ratolí, el lector de discos òptics —CD-ROM (Compact Disc Read Only Memory) i DVD (Digital Video Disc o Digital Versatile Disc)—, un micròfon, un escàner, un lector de codis de barres, etc.;
- els *de sortida* mostren informació produïda pel processador: el monitor o pantalla, un canó de projecció, una impressora, uns altaveus, etc., i
- els d'*entrada/sortida* permeten transferir informació en ambdós sentits: *discos magnètics* durs, interns (fixos) o externs, discos magnètics flexibles (*floppy disks*) o *disquets*, *gravadores* de discos òptics (CD-R, CD-RW, DVD-R, DVD-RW, DVD+R, DVD+RW, etc.), *dispositius USB de memòria flaix* (anomenats també *pen drive*, *USB flash drive*, *memòria USB* o *reproductor MP3* si permeten escoltar música comprimida amb aquest format), *mòdems* o *routers* per connectar l'ordinador a una línia telefònica, etc.

El principal mitjà d'emmagatzematge no volàtil d'informació solen ser els discos magnètics i, per això, una característica important d'un ordinador és la capacitat del seu disc dur. Aquesta capacitat es pot ampliar substituint el disc o afegint-hi discos addicionals. L'accés a disc (o a altres perifèrics) és molt més lent que l'accés a la memòria RAM, per la qual cosa se sol intercalar una petita memòria ràpida —un *buffer* o *cache de disc*— que acumula blocs d'informació que s'ha d'escriure o llegir abans de transferir-los entre el disc i la memòria RAM.

Els perifèrics s'acoblen a l'ordinador a través de *connexions* que també s'anomenen *ports, sortides* o *entrades*, com ara *USB*, *FireWire* (anomenat també *IEEE 1394* o *iLink* en les videocàmeres), *ATA* o *IDE*, *Serial ATA*, *SCSI*, *paral·lel*, *sèrie*, etc. Les *connexions a xarxa* permeten que l'ordinador es comuniqui amb altres ordinadors. La més utilitzada avui dia s'anomena *Ethernet*. Si no disposem de línia Ethernet, necessitarem un mòdem per transformar la informació que ha de sortir de l'ordinador o entrar-hi en senyals elèctrics transmissibles a través d'una línia telefònica (mòdems *RTB* o *ADSL*),

[2] Aquesta denominació és poc afortunada: de fet, tant la memòria RAM com la ROM són d'*accés aleatori* (Random Access), en el sentit que podem accedir a qualsevol dada emmagatzemada de manera directa (sense haver de llegir les dades anteriors).

[3] La paraula *xip* se sol aplicar a qualsevol component microelectrònic amb circuits integrats i moltes connexions elèctriques.

o de la xarxa elèctrica (mòdem *PLC*). La comunicació amb altres ordinadors o amb perifèrics es pot fer també sense fils (mitjançant ones electromagnètiques), fent servir les tecnologies *Bluetooth*, *Wi-Fi* o *Airport*, etc. El nombre de connexions d'un ordinador es pot ampliar si aquest disposa de *ranures d'expansió* o *slots* (*PCI*, *PCMCIA*, etc.), que són espais preparats per introduir-hi *targetes* (plaques amb components electrònics) que poden incorporar noves connexions. Aquestes targetes poden realitzar certes operacions, com ara la conversió d'un senyal *analògic* (que varia de forma contínua) produït per un aparell en un senyal *digital* (expressat en forma numèrica) intel·ligible per l'ordinador.

La torre conté, normalment, una placa amb un circuit imprès —la *placa base*— en què estan endollats el processador, les memòries ROM i RAM, el disc dur, la disquetera, el lector i/o la gravadora de CD/DVD, diversos ports i ranures d'expansió, targetes gràfica[4] i de so, una font d'alimentació, etc. En els ordinadors compactes tot això va integrat amb el monitor i els altaveus, i en els portàtils s'hi afegeix també el teclat i una mena de ratolí.

Exercici 1.1
Classifiqueu els perifèrics següents com d'entrada, de sortida o d'entrada/sortida: a) disc magnètic; b) impressora; c) teclat; d) monitor; e) mòdem.
Solució: a) E/S; b) S; c) E; d) S (si no és tàctil); e) E/S.

1.3 Codificació de la informació

Els ordinadors treballen amb informació codificada en forma *digital* —és a dir, numèrica— tant si es tracta de xifres com si són lletres, imatges, sons, etc. Així, una imatge es pot codificar en format digital (*digitalitzar*) fent servir un escàner, que la divideix en una quadrícula molt fina i emmagatzema el color de cada quadradet (*píxel*) mitjançant tres nombres naturals que representen les intensitats de tres colors primaris; per exemple el vermell, el verd i el blau (*RGB*: Red, Green and Blue).

Per a la major part de les aplicacions, l'usuari d'un ordinador no necessita saber com es codifica la informació que introdueix o que extreu de l'ordinador, ja que els perifèrics de sortida (monitor, impressora...) li presenten descodificada i els d'entrada (teclat, ratolí...) la codifiquen de manera automàtica. Això no obstant, és interessant tenir una idea de com es fa aquesta codificació.

La codificació que fan servir actualment els ordinadors és de tipus *binari*, la qual cosa vol dir que utilitzen només dos dígits: el 0 i l'1. D'una manera molt simplista, podríem imaginar la memòria RAM com un taulell amb moltes bombetes, cada una de les quals representa un 0 o un 1 segons estigui apagada o encesa. D'acord amb aquest símil, la CPU aniria encenent i apagant bombetes quan ha de guardar informació en la memòria, i per llegir aquesta informació haurà de comprovar l'estat —apagat o encès— de les bombetes. És evident que un sistema com aquest només permet emmagatzemar informació codificada amb un sistema binari, i que la informació es perdrà si s'interromp el subministrament elèctric i no ha estat prèviament copiada en un suport no volàtil; per exemple, un disc magnètic. En aquest, un 0 o un 1 es podrien enregistrar magnetitzant una petita superfície (un "punt") del disc en una direcció o en l'oposada, respectivament. Aquesta magnetització no es perd quan s'apaga l'ordinador, de manera que les dades enregistrades es conserven. Per llegir després la informació caldrà anar detectant el sentit de la magnetització a cada punt del disc.

Veiem com s'expressa un nombre natural qualsevol en codificació o notació binària. Per expressar un nombre natural en notació decimal —la qual s'identificarà afegint-hi la indicació $)_{10}$— escrivim els dígits decimals (0, 1, 2, 3, 4, 5, 6, 7, 8 o 9) que, multiplicats per successives potències de 10 i sumats, donen aquest nombre:

$$3504)_{10} = 3 \times 10^3 + 5 \times 10^2 + 0 \times 10^1 + 4 \times 10^0 = 3000 + 500 + 0 + 4$$

Anàlogament, la notació binària d'un nombre natural consisteix en una successió de dígits binaris (0 o 1) que, multiplicats per successives potències de 2 i sumats, donen el nombre representat. Per exemple, la notació binària 1101 representa el nombre decimal 13:

$$1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 0 + 1 = 13)_{10}$$

[4]La presentació d'imatges al monitor sol estar gestionada per un processador dedicat exclusivament a aquesta tasca (*GPU*: Graphics Processing Unit), el qual pot estar a la placa base o a la targeta gràfica.

Exercici 1.2
Indiqueu els nombres naturals en codificació decimal que corresponen a les notacions binàries $1010)_2$, $1000)_2$, $1111101000)_2$ i $10000000000)_2$.
Solució: $10)_{10}$, $8)_{10}$, $1000)_{10}$ i $1024)_{10}$.

Per expressar un nombre decimal en notació binària es divideix successivament per 2 fins a obtenir un quocient més petit que 2. Aquest quocient, seguit de tots els residus de les divisions anteriors (cada un dels quals pot ser 0 o 1) començant per l'últim, constitueixen la notació binària d'aquell nombre. Per exemple, per expressar el $13)_{10}$ en notació binària, farem les divisions:

$$\begin{array}{rrrr} 13 & \boxed{2} & & \\ 1 & 6 & \boxed{2} & \\ & 0 & 3 & \boxed{2} \\ & & 1 & 1 \end{array}$$

que condueixen al resultat $1101)_2$.

Exercici 1.3
Escriviu en notació binària els nombres $63)_{10}$, $64)_{10}$ i $65)_{10}$.
Solució: $111111)_2$, $1000000)_2$, $1000001)_2$.

La unitat d'informació més petita que pot manipular un ordinador que treballa amb codificació binària és un dígit binari o *bit* (B-inary dig-IT). Qualsevol memòria de l'ordinador està estructurada com una successió de bits, cadascun dels quals podrà emmagatzemar un 0 o un 1. Com que el bit és una unitat de mesura molt petita, s'ha definit l'*octet* o *byte*, que equival a 8 bits, i se simbolitza mitjançant una 'B'. Els múltiples decimals del bit o del byte es poden identificar amb els mateixos prefixos que s'utilitzen per a qualsevol altra unitat (kilo, mega, giga, tera, etc.). No obstant això, el fet que els ordinadors utilitzin codificació binària fa que sigui convenient utilitzar múltiples en què el factor sigui una potència de 2 en lloc de 10. Això s'explica perquè les potències de 2 són nombres rodons en notació binària, de la mateixa manera que les de 10 ho són en la notació decimal; per exemple, $2^6 = 64)_{10} = 1000000)_2$ (exercici 1.3). Com que la potència de 2 més propera a 1000 és $2^{10} = 1024)_{10} = 10000000000)_2$ (exercici 1.2), se sol utilitzar aquest nombre en lloc de 1000 per definir els múltiples kilo, mega, etc., de la mateixa manera que s'ha utilitzat un nombre rodó en binari, $1000)_2 = 8)_{10}$, en comptes del nombre rodó en decimal $10)_{10} = 1010)_2$ per definir el byte. Això ha creat una ambigüitat en el significat d'aquests prefixos que ha estat resolta per la Comissió Electrotècnica Internacional el desembre de 1998 mitjançant la introducció de prefixos específics per als múltiples binaris. La taula següent recull aquests noms juntament amb els factors corresponents, els símbols que els representen i els prefixos decimals dels quals deriven:

Factor binari	**Nom**	**Símbol**	**Factor decimal**	**Nom**	**Símbol**
1024^1	kibi	Ki	1000^1	kilo	k
1024^2	mebi	Mi	1000^2	mega	M
1024^3	gibi	Gi	1000^3	giga	G
1024^4	tebi	Ti	1000^4	tera	T
1024^5	pebi	Pi	1000^5	peta	P
1024^6	exbi	Ei	1000^6	exa	E

Els prefixos binaris encara s'utilitzen poc i, pel que fa a la memòria RAM, la major part de la gent fa servir les unitats kB, MB, GB, etc. per referir-se als KiB, MiB, GiB, etc. Ara bé, en el cas particular dels discs durs, la informació comercial sol expressar la seva capacitat en GB o TB decimals. Per exemple, si la capsa d'un disc dur indica que aquest és de 100 GB, en consultar la seva capacitat a través de l'ordinador podem trobar-nos que aquest indica un valor d'uns 93,1 GB, quan hauria de dir

93,1 GiB (gibibytes) (vegeu l'exercici 1.4).

Exercici 1.4
Quina capacitat en GiB té un disc de 100 GB?
Solució: uns 93,132 GiB.

Exercici 1.5
Quanta memòria, expressada en KiB, ocupa una imatge de 640 × 480 píxels si es fan servir 8 bits per codificar la intensitat de cada un dels tres colors primaris (imatge de 24 bits)?
Solució: 900 KiB.

En contextos tècnics també s'utilitza, de vegades, la unitat *paraula* per mesurar quantitats d'informació. La paraula ve a ser la unitat d'informació amb què treballa el processador, i pot ser de 32, 64... bits, depenent del tipus de processador. Els múltiples de la paraula es defineixen de la mateixa manera que els dels bytes o els bits; així, una mebiparaula —en anglès *mebiword* (MiW)— són 1024^2 paraules. La posició de cada paraula s'identifica mitjançant un nombre que es coneix com l'*adreça* de la posició de memòria. Per exemple, una representació esquemàtica de la memòria RAM d'un ordinador amb paraules de 32 bits podria ser:

Adreça	**Contingut de la paraula de memòria**
0	01000101 01001101 00010010 01001010
1	10100001 01010011 01000101 01001101
2	00010010 01001010 00010010 01001010
...	...

La velocitat de transmissió d'informació a través d'una xarxa (Ethernet, línia telefònica, etc.) sol expressar-se en múltiples de bit per segon; així, en el moment d'escriure aquest text, la xarxa telefònica admet una velocitat de transmissió d'informació de 56 Kibit/s (kibibits per segon) —és a dir, 56 × 1024 bit/s— si s'utilitza un mòdem convencional i de 20 Mibit/s (mebibits per segon) amb el sistema ADSL (Asymmetric Digital Subscriber Line); les targetes d'Ethernet arriben fins a 10 Gibit/s.[5] En la pràctica, les velocitats efectives solen ser més petites que aquests valors màxims (almenys un 20%), entre d'altres coses, perquè l'enorme flux d'informació (incloent-hi els protocols de comunicació entre ordinadors) satura tant les línies per les quals es transmet com els ordinadors que la distribueixen (els *servidors*).

Exercici 1.6
Indiqueu el temps que trigarà (com a mínim) a transmetre's 700 MiB d'informació a través de la xarxa telefònica si fem servir un mòdem de 56 Kibit/s. ¿I si fem servir una línia ADSL que permet *baixar* informació a 3 Mibit/s i *pujar-la* a 320 Kibit/s (el lector haurà endevinat per què se'n diu "asimètric" aquest sistema de comunicació)?
Solució aproximada: 28 h; 31 min; 5 h.

Per codificar nombres enters (inclosos els negatius) se sol reservar un bit per al signe (una manera de fer-ho és posar-hi un 0 si el nombre és positiu i un 1, si és negatiu). Per codificar nombres amb decimals es guarden les seves xifres significatives per una banda i la posició de la coma per l'altra. El signe s'indica en un altre bit. Per codificar lletres, xifres (com a caràcters, no com a nombres) i altres símbols ortogràfics, matemàtics, etc. es fa servir normalment la codificació ASCII (American Standard Code for Information Interchange) —també anomenada US-ASCII—, que atribueix una

[5] Aquestes xifres canvien molt ràpidament i és probable que estiguin obsoletes quan el lector llegeixi aquest text. D'altra banda, no sempre és clar si les dades comercials estan expressades amb múltiples decimals, binaris o, fins i tot, barreges dels dos tipus.

successió de 7 bits a cada caràcter. A continuació mostrem alguns exemples d'aquesta codificació:

Caràcter	Codificació US-ASCII
A	$1000001)_2 = 65)_{10}$
a	$1100001)_2 = 97)_{10}$
2	$0110010)_2 = 50)_{10}$
+	$0101011)_2 = 43)_{10}$

D'aquesta manera es poden codificar 128 caràcters (vegeu l'exercici 1.7).[6] Com succeeix amb molts "estàndards", hi ha diverses variants i extensions de la codificació US-ASCII. Entre les més utilitzades hi ha les extensions ISO/IEC 8859-1 (o Latin-1) i la ISO/IEC 8859-15, que afegeixen un bit a la US-ASCII per tal de duplicar el nombre de codis (vegeu l'exercici 1.7) i poder incloure lletres accentuades i certs caràcters que no existeixen en anglès.[7]

Exercici 1.7
Quants caràcters es poden codificar mitjançant la codificació ASCII original (de 7 bits per caràcter)? I si fem servir una codificació ASCII estesa (de 8 bits per caràcter)?
Solució: $2^7 = 128$; $2^8 = 256$.

Exercici 1.8
Quants nombres enters podem guardar (en format binari) en 4 bytes?
Suggeriment: Tingueu en compte que, en codificació decimal, tres dígits permeten codificar des del 0 fins al 999; és a dir, 10^3 nombres.
Solució: $2^{32} = 4\,294\,967\,296$.

Exercici 1.9
Quantes pàgines de text es poden emmagatzemar (en codificació ASCII de 8 bits per caràcter) en un CD-ROM de 700 MiB? (suposeu que hi ha, per terme mitjà, 32 línies per pàgina i 64 caràcters per línia).
Solució: 358 400 pàgines.

Exercici 1.10
Indiqueu si és certa o falsa cadascuna de las afirmacions següents: a) Tota la informació que s'emmagatzema en un ordinador està codificada amb els dígits 0 i 1. b) La informació continguda en la memòria ROM es perd quan es tanca l'ordinador. c) La informació que es troba en la memòria RAM es pot esborrar i modificar. d) El monitor d'un ordinador forma part del seu software.
Solució: a) Certa; b) falsa; c) certa; d) falsa.

1.4 El programari

D'una manera molt general, podem classificar el *software* o programari en diferents categories: el sistema operatiu, les aplicacions i, si adoptem l'accepció més amplia del terme *software*, els documents.

El *sistema operatiu* és un conjunt d'instruccions que permeten començar a treballar amb l'ordinador i fer les tasques més generals, com llegir el contingut d'un CD, copiar-ne una part en el disc fix,

[6] De fet, una part de les 128 combinacions de bits (les que corresponen als nombres 0 a $31)_{10}$ i $127)_{10}$) estan reservades per a caràcters de control no imprimibles, els quals s'utilitzen per controlar certs dispositius; per exemple, impressores. Així, el caràcter $10)_{10}$ —*line feed*— significa canvi de línia. El lector interessat en els diferents estàndards de codificació de caràcters pot consultar la web: *http://es.wikipedia.org/wiki/ASCII*

[7] Tot i així, la gran varietat de caràcters que s'utilitzen en les diferents llengües del món fa que les 256 possibilitats que ofereixen 8 bits es quedin molt curtes, i s'estan desenvolupant codificacions que pretenen ser pràcticament exhaustives. La més estesa és la UNICODE, que admet 1 114 112 ($= 2^{20} + 2^{16}$) caràcters, dels quals a hores d'ara estan assignats uns 96 000. Els primers 256 coincideixen amb la codificació ISO/IEC 8859-1, els 128 primers de la qual són els mateixos que els de la US-ASCII.

imprimir una imatge en una impressora, mostrar el contingut d'un dispositiu de memòria flaix, etc. El caràcter general d'aquestes instruccions fa que sigui un programari necessari independentment de la utilització concreta que es vulgui fer de l'ordinador. Hi ha sistemes operatius per a cada tipus de processador, i pot haver-n'hi més d'un que funcioni en un ordinador determinat. Per exemple, un ordinador amb un processador Intel Pentium pot funcionar amb diferents versions dels sistemes operatius Windows, GNU/Linux i MacOS X, entre d'altres.

Una *aplicació* o *programa* és un conjunt d'instruccions que permet realitzar tasques lligades a un ús concret de l'ordinador. Per exemple, un *editor* o *processador de textos* és una aplicació que permet escriure textos, guardar-los en discos, llegir-los i modificar-los, imprimir-los, etc.; un *full de càlcul* és una aplicació que permet fer diverses operacions amb col·leccions de dades disposades en forma de taula; un *gestor de bases de dades* és una aplicació que permet manipular la informació d'un banc de dades (ordenar-la, classificar-la, seleccionar-ne un part, etc.); un *navegador* és una aplicació que permet *navegar* per la xarxa Internet (cercar informació, visualitzar-la, transferir-la al disc fix, etc.); i els *compiladors* i *intèrprets* són aplicacions que tradueixen un conjunt d'instruccions escrit en un llenguatge de programació estàndard a un llenguatge propi de l'ordinador, tal com veurem en el capítol 2.

Determinades aplicacions permeten crear *documents* que contenen informació. És el cas dels editors de textos, que creen i modifiquen documents textuals (cartes, informes, etc.); els fulls de càlcul, que creen i modifiquen taules de valors que poden estar interrelacionats (documents que s'anomenen també *fulls de càlcul*); els gestors de bases de dades, que creen o modifiquen bases de dades; etc.

La classificació anterior no és estricta. Així, les aplicacions que s'instal·len en l'ordinador juntament amb el sistema operatiu —com ara una aplicació per navegar per Internet, un visualitzador de vídeo o un reproductor de so— i els documents d'informació i ajuda poden considerar-se part del sistema.

1.5 Sistemes operatius

Quasi tots els sistemes operatius actuals permeten manipular l'ordinador de dues maneres:

- En *mode gràfic* o *amb interfície gràfica* (GUI: Graphical User Interface): el monitor presenta una sèrie de *menús* desplegables on les instruccions del sistema operatiu apareixen escrites i se seleccionen amb l'ajuda d'un ratolí (*clicant* sobre un menú i *arrossegant* el ratolí fins que *s'activi* la instrucció desitjada), *finestres* (rectangles dibuixats sobre la pantalla) que poden contenir gràfics i *botons clicables* per escollir certes opcions, dibuixets anomenats *icones* per representar certs components de maquinari o de programari, etc.

- En *mode text*: cada instrucció s'ha de teclejar i s'executa en prémer la tecla de retorn (⏎), anomenada també *Entrar* o *Intro*. El monitor —o una finestra d'aquest on apareix el text teclejat— pot mostrar caràcters però no gràfics. Se'n sol dir una *consola* o *terminal*.

El mode text presenta certs inconvenients: dificultat de memoritzar les instruccions menys usuals, lentitud quan s'han d'escriure instruccions llargues, facilitat de cometre errors en teclejar-les, etc. El mode gràfic resulta més intuïtiu i fàcil d'aprendre, cosa que redueix considerablement les necessitats de memorització i les possibilitats d'errors. Els sistemes més utilitzats en l'actualitat són el Windows i les diverses derivacions de l'Unix: GNU/Linux, Mac OS X, Solaris, etc. El mode gràfic és, amb diferència, la manera d'operar més estesa, i el treball en mode text està cada dia més restringit als desenvolupadors de programari i als encarregats del manteniment de sistemes.

Als apèndixs A i B podeu trobar una breu ressenya sobre el desenvolupament històric dels sistemes operatius i, en particular, de les interfícies gràfiques.

1.6 Estructuració dels discos

De la mateixa manera que es necessita un cert grau d'ordre en els documents que tenim al nostre despatx per poder localitzar-los amb facilitat, és convenient estructurar la informació guardada en el disc fix —o en qualsevol altre mitjà d'emmagatzematge d'informació— d'un ordinador. Una bona

part de les instruccions dels sistemes operatius serveixen per crear aquesta estructuració i treure'n profit.

Un *fitxer* o *arxiu* és un bloc d'informació que s'identifica mitjançant un nom. La informació continguda en un fitxer pot ser tant una part del sistema operatiu, com una aplicació (o una part d'aquesta) o un document. Així, quan escrivim un document amb un editor de textos i el guardem en el disc dur donant-li un nom, estem creant al disc un fitxer amb aquest nom.[8] El mateix editor de textos estarà també emmagatzemat en un fitxer del disc (o en més d'un). Els fitxers poden contenir també dades per efectuar un càlcul matemàtic, els resultats d'aquest, l'aplicació que l'efectua, imatges o so digitalitzats, etc.

En el sistema operatiu DOS, antecessor del Windows, així com en les primeres versions d'aquest, els noms de fitxer poden tenir, com a màxim, vuit caràcters i, opcionalment, un punt i tres caràcters més. Aquests últims s'anomenen l'extensió del nom, i se solen utilitzar per indicar el tipus d'informació continguda al fitxer. Els caràcters poden ser lletres, xifres i caràcters especials excepte `/ \ [ ] : ; < > + = . ,`. Per exemple, un fitxer que conté una aplicació per calcular el producte de dues matrius es podria anomenar `prod2m.exe`, on l'extensió `exe` indicaria que el fitxer conté una aplicació; és a dir, que es tracta d'un fitxer *executable*. Les dades que necessita l'aplicació, és a dir, les matrius a multiplicar es poden introduir amb el teclat, però també es poden llegir d'un fitxer on estiguin guardades, que es podria anomenar `prod2m.dad`. El resultat del càlcul —la matriu producte— es pot mostrar al monitor, escriure en una impressora o guardar en un fitxer del disc amb el nom de, per exemple, `prod2m.res`.

En la major part dels sistemes operatius actuals (Windows, GNU/Linux, Mac OS X...) hi ha poques restriccions en els noms que es poden donar als fitxers. Així, el fitxer de dades considerat abans es podria dir, en qualsevol d'aquests sistemes, `Matrius a multiplicar`. No obstant això, és convenient no utilitzar caràcters que tinguin un significat especial per al sistema, com `/`, `\`, `-` o un punt com a primer caràcter, ni espais en blanc.[9] D'altra banda, encara hi ha aplicacions per al Windows que només funcionen correctament si els noms dels documents de treball que empren s'ajusten al sistema de 8+3 caràcters abans esmentat.

Un disc pot contenir una sèrie de fitxers sense cap altra estructuració i, de fet, aquesta és la manera habitual de disposar els fitxers en un disquet. No obstant això, quan el nombre de fitxers és considerable —com succeeix en el disc fix— és convenient agrupar-los per temàtiques en diferents *carpetes* o *directoris*, com ho faríem amb els papers del despatx. Així, una carpeta pot contenir els fitxers que constitueixen el sistema operatiu, una altra les aplicacions, una altra els documents creats per l'usuari, etc. Els noms de les carpetes han d'ajustar-se a les mateixes especificacions que els dels fitxers. Per identificar un fitxer que es troba en una carpeta determinada s'escriu el nom d'aquesta i, a continuació, el del fitxer separat pel caràcter `\` (*barra inversa*) en el sistema Windows i per `/` (*barra inclinada*) en els altres sistemes esmentats. Per exemple, en Windows, la identificació completa d'un fitxer amb nom `apunts_mates` que es troba en la carpeta `documents` del disc dur —el qual s'identifica, normalment, mitjançant la notació `C:`—, serà `C:\documents\apunts_mates`. Als altres sistemes la identificació seria `/documents/apunts_mates` . En endavant ens referirem a sistemes de la família Unix, inclosos el Linux i el Mac OS X, sempre que no indiquem el contrari. Per aplicar la discussió al sistema Windows s'haurien de substituir les barres inclinades (`/`) per barres inverses (`\`) i afegir a l'esquerra la identificació del disc: `A:` per a la disquetera i `C:`, `D:`... per als discs durs, CD, etc.

Cada carpeta o directori es pot estructurar en noves *carpetes* o *subdirectoris*. Per exemple, podríem definir, en el directori `documents`, dos subdirectoris: `dibuixos` i `apunts` i ens referirem a un document d'aquest últim com `/documents/apunts/apunts_mates`. Les paraules carpeta, directori i subdirectori s'utilitzen indistintament (tot directori és subdirectori d'algun altre o del disc sencer). La indicació completa del directori on es troba un fitxer s'anomena el *camí* (*path*) o *ubicació* del fitxer. Així, en l'exemple anterior `/documents/apunts/` és el camí que condueix al fitxer `apunts_mates`. Els directoris confereixen al disc una estructura d'arbre, en què les branques serien els directoris i les fulles els fitxers (figura 1.2). El nivell més baix d'aquesta estructura se sol anomenar l'*arrel* (`/`)

[8] La denominació *fitxer* o *arxiu* es remunta als orígens de la informàtica: un fitxer es podia emmagatzemar com un conjunt de fitxes perforades, que eren fitxes de cartolina sobre les quals s'enregistrava la informació mitjançant una perforadora. Cada fitxa contenia una línia del fitxer, amb un màxim de 80 caràcters. La perforadora s'assemblava a una màquina d'escriure, i escrivia els caràcters de la línia a la part superior de la fitxa al temps que feia uns forats a sota cada caràcter; aquests permetien a la lectora de fitxes —un perifèric d'entrada— identificar els caràcters i transmetre la informació al processador.

[9] En certs casos s'han d'escriure entre cometes els noms de fitxer que contenen caràcters d'aquests.

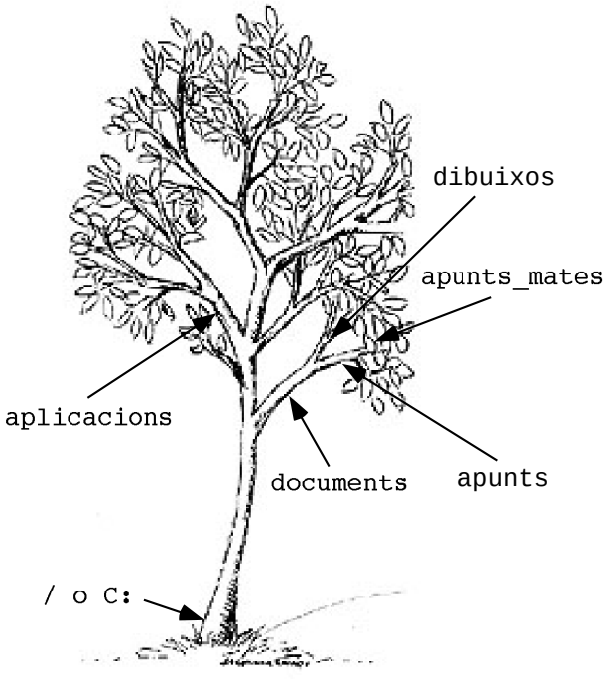

Figura 1.2: Estructuració d'un disc.

(s'hauria de parlar d'un arbust, més que d'un arbre, ja que no hi ha tronc). Es diu que els fitxers i subdirectoris *pengen* del directori on s'ubiquen.

1.7 El sistema operatiu GNU/Linux

El GNU/Linux és un sistema operatiu *lliure* (vegeu l'apèndix B) que es pot obtenir gratuïtament a la xarxa Internet; per exemple, de l'organisme sense ànim de lucre Debian, o de l'empresa Canonical Ltd. (distribució Ubuntu, basada en Debian). També es pot comprar —a un preu força moderat— a empreses que recopilen el que consideren més important i el distribueixen (RedHat, SuSe, Mandriva ...). A més, aquestes proporcionen suport tècnic durant un cert període a partir de la data de compra. Les distribucions de GNU/Linux inclouen diferents GUI (KDE, Gnome, etc.) que faciliten les operacions més usuals; no obstant això, és interessant saber com es poden escriure i executar les instruccions del sistema, ja que s'ha de treballar en mode text per poder treure-li tot el suc. Les diferents GUI funcionen de manera bastant semblant. No obstant això, s'ha de tenir en compte que, fins i tot per a una mateixa distribució i GUI, poden haver-hi petites diferències depenent de la versió i de com s'hagi configurat el sistema.

Quan s'engega un ordinador, aquest busca un sistema operatiu en els discos accessibles i, si el troba, comença a llegir les seves instruccions d'arrencada i copiar-les o *carregar-les* en la memòria RAM. Si tenim més d'un sistema operatiu instal·lat al disc dur, per exemple, Windows i GNU/Linux, trobarem de seguida una pantalla que ens oferirà la possibilitat d'escollir el sistema que vulguem utilitzar. Seleccionarem l'opció Linux amb les *tecles del cursor* (que estan a la part inferior, entre el teclat alfanumèric i el numèric) i, en prémer la tecla de retorn (⏎), es carregarà aquest sistema.

El sistema GNU/Linux també es pot carregar des d'un disc compacte d'arrencada o *autònom* (un *live CD*), la qual cosa permet utilitzar-lo sense haver d'instal·lar res al disc dur. Només caldrà engegar el PC amb el CD autònom ficat en el lector de CD i, si la BIOS (Basic Input/Output System) està configurada de la manera adient,[10] arrencarà el sistema del CD. En el moment d'escriure aquest text, els CD autònoms més utilitzats deriven d'un que s'anomena *Knoppix* (vegeu la bibliografia).

[10]Si l'ordinador busca el sistema operatiu en el disc dur abans que en el CD s'haurà de modificar la BIOS per invertir l'ordre de cerca. Per modificar la BIOS s'ha d'interrompre l'arrencada (quan s'inicia) prement una tecla que pot ser Supr, F2, F1, Esc, etc., depenent del fabricant de la placa base. En començar l'arrencada sol aparèixer (de manera força fugaç) una indicació de la manera d'entrar en la BIOS.

El GNU/Linux és un sistema *multiusuari*, és a dir, permet definir diferents usuaris, cada un amb el seu nom o *codi* i la seva *paraula de pas* (*password*), que poden utilitzar el mateix ordinador —en moments diferents o, fins i tot, simultàniament si hi ha altres ordinadors que hi estan connectats via xarxa— sense perill d'interferències: cada usuari té assignat un *directori propi* (el *directori* o *carpeta de l'usuari*), pot protegir els seus fitxers perquè no puguin ser esborrats, modificats o, fins i tot, vists per un altre, pot configurar l'entorn de treball al seu gust, etc. Sempre hi ha un usuari definit —el *root*— que té accés a tota la informació i pot executar qualsevol instrucció. Els altres usuaris tenen algunes opcions vetades per tal de garantir la integritat del sistema envers errors accidentals o accions malintencionades.

1.7.1 La GUI

La resta d'aquest capítol està concebuda per a un lector que disposi d'un ordinador arrencat amb el sistema operatiu GNU/Linux per tal de posar en pràctica, a mesura que llegeix el text, les instruccions que s'aniran introduint.[11] Per iniciar una sessió de treball s'ha d'introduir, normalment, un codi i una paraula de pas prèviament donats d'alta per l'usuari *root*. En endavant suposarem que el codi de l'usuari és `usuari1` i que el seu directori és `/home/usuari1/`. Una vegada iniciada la sessió, el monitor mostra algunes icones distribuïdes per la pantalla —l'*escriptori*— amb el nom corresponent a sota: pot haver-hi una paperera, un disquet o *floppy*, un disc dur etiquetat com a *hda1*, etc. A la part inferior (de vegades a la superior) hi ha una franja amb més icones (la *barra de menús*). Una d'aquestes sol ser una caseta que representa la carpeta de l'usuari (la seva *home*). Clicant-hi a sobre una sola vegada amb el botó esquerre del ratolí s'obre una finestra que mostra el contingut de la carpeta; és a dir, els noms dels fitxers que conté i els dels subdirectoris (o carpetes) que hi pengen.

Una de les aplicacions incloses amb el sistema és un editor de textos senzill que ens serà molt útil per crear i modificar fitxers amb programes. Pot ser que hi hagi una icona en l'escriptori o en la barra de menús que representi aquest editor (un llapis, una llibreta, un paper amb una ploma, etc.); en aquest cas, *obrirem* l'editor clicant sobre la icona. Si no, haurem de desplegar el *menú d'aplicacions* (sol estar a la part esquerra de la pantalla), buscar un editor i seleccionar-lo. En obrir-se o *arrencar-se*, l'editor es llegeix del disc i es copia o *carrega* en la memòria RAM perquè puguem treballar-hi. Un cop carregat, apareix una finestra en blanc que representa un fitxer nou. Si teclegem un text, aquest apareixerà en la finestra just a continuació del *cursor* (la ratlleta vertical que parpelleja). El text que escrivim es va guardant a la memòria RAM i, en conseqüència, es perdrà si, per exemple, es produeix un tall de corrent. Per guardar-lo en una memòria permanent (disc dur, disquet, memòria USB, etc.) hem de seleccionar l'opció *Desar* o *Guardar* del menú *Fitxer* o *Arxiu*, a la part superior de la finestra de l'editor. En fer-ho, s'obre una finestra perquè hi escrivim el nom que vulguem donar al fitxer. A la part superior d'aquesta finestra s'indica la ubicació que tindrà el fitxer, la qual es pot modificar clicant-hi i escrivint la nova ubicació. En clicar sobre el botó *Acceptar* o *Desar* , el fitxer es gravarà al disc, com podrem comprovar clicant sobre la finestra de la carpeta de l'usuari i buscant-lo. Si tornem a l'editor (clicant sobre la seva finestra) i seguim escrivint o modifiquem el text que hi havia, les modificacions afectaran només la versió del fitxer que resideix a la memòria RAM, la qual no es guardarà al disc fins que tornem a seleccionar l'opció *Desar*. Ara ja no caldrà introduir cap nom, ja que es mantindrà el que vam assignar la primera vegada. Si volem conservar la versió del fitxer que hi havia al disc i crear-ne un de nou amb la versió modificada que resideix a la memòria RAM, haurem de seleccionar l'opció *Desar com...*, que ens demanarà el nom que volem donar al nou fitxer.[12]

Convé dedicar un cert temps a escriure sobre el fitxer i fer diferents modificacions del text escrit per familiaritzar-se amb el funcionament de l'editor: clicar sobre el text escrit per insertar-hi una paraula, clicar sobre el text i arrossegar el ratolí per seleccionar un fragment que vulguem esborrar (tecla [←]), copiar (opció *Copiar* del menú *Editar* i, després de clicar en un altre punt del text, opció *Enganxar*), moure (opcions *Tallar* i *Enganxar*), insertar línies en blanc amb la tecla de retorn ([↩]), moure's pel text fent servir les tecles del cursor, etc.

[11] S'ha de tenir present que poden haver-hi moltes variacions en els detalls de funcionament i aspecte de la GUI —depenent, sobretot, de la distribució de GNU/Linux emprada— respecte del que descrivim en aquest apartat.

[12] Si s'està treballant amb el sistema operatiu arrencat des d'un *live CD* el directori de l'usuari està en un *disc virtual* o *ramdisk*, que és una zona de la memòria RAM que es presenta com si fos un disc. Per tant, qualsevol fitxer guardat en aquest directori que es vulgui conservar després de tancar la sessió de treball s'haurà de copiar prèviament en una memòria permanent.

Si tornem a la carpeta de l'usuari podrem eliminar un dels fitxers que hem creat arrossegant-lo amb el ratolí fins a la icona de la paperera. En realitat, aquesta operació no elimina definitivament el fitxer, que encara es pot recuperar clicant sobre la paperera i arrossegant-lo de nou sobre la carpeta de l'usuari (o qualsevol altre punt de l'escriptori). Per esborrar-lo definitivament hauríem de buidar la paperera, després de llençar-hi el fitxer, clicant-hi amb el botó dret del ratolí i seleccionar l'opció *Buidar la paperera*. També es pot eliminar definitivament un fitxer seleccionant l'opció *Esborrar* en el menú que apareix en clicar amb el botó dret del ratolí sobre la icona del fitxer.

Si, al menú *Editar* de la carpeta de l'usuari, seleccionem l'opció *Crear nou* seguida de *Carpeta...* i introduïm un nom en la finestra que apareix —per exemple, `exercicis`— haurem creat un subdirectori del directori de l'usuari que es representarà mitjançant una nova icona amb forma de carpeta. Clicant sobre aquesta icona s'obrirà una finestra en blanc, ja que el nou subdirectori no conté cap fitxer ni carpeta (està buit). La fletxa que apunta cap amunt a la part superior d'aquesta finestra ens tornarà a la carpeta de l'usuari. Des d'aquesta carpeta podem arrossegar un dels fitxers que conté a la nova carpeta `exercicis`, la qual cosa farà aparèixer un petit menú: si escollim l'opció *Moure* el fitxer canviarà d'ubicació (desapareixerà d'`usuari1` i passarà a `exercicis`) i si escollim *Copiar* es mantindrà a la ubicació original i n'apareixerà un duplicat a la carpeta `exercicis`.

Per copiar fitxers a un disquet o des d'un disquet hem d'introduir-lo en la disquetera —amb el foradet de la cantonada superior esquerra tancat si volem escriure-hi—, clicar sobre la icona amb forma de disquet i esperar que s'obri una finestra que en mostrarà el contingut.[13] Llavors, el disquet estarà disposat per treballar-hi (*muntat*), la qual cosa s'indica mitjançant un petit triangle al costat de la icona. Si volem copiar fitxers al disquet o des d'aquest al disc fix els haurem d'arrossegar de la mateixa manera que quan els copiem entre diferents carpetes d'un mateix disc. A diferència del que succeeix al sistema operatiu Windows, no hem de prémer directament el botó d'expulsió del disquet quan vulguem treure'l després de guardar-hi informació; abans hem de *desmuntar-lo*, per la qual cosa tancarem la finestra del disquet, clicarem sobre la seva icona amb el botó dret del ratolí i seleccionarem l'opció *Desmuntar*. A més, s'ha d'esperar que s'apagui el llumet que hi ha al costat del botó d'expulsió, indicatiu d'activitat a la disquetera (això també s'aplica al sistema Windows).

Quan endollem un dispositiu USB o FireWire (per exemple, un dispositiu de memòria flaix o un disc dur extern) apareix una icona en l'escriptori que el representa. Amb aquests dispositius es treballa de manera similar que amb els disquets: en clicar sobre la icona del dispositiu aquest es munta per poder treballar-hi, i abans de desendollar-lo s'ha de desmuntar clicant sobre la icona amb el botó dret del ratolí i seleccionant l'opció *Desmuntar* en el menú que hi apareix (el muntatge s'efectua automàticament si està activada l'opció corresponent del sistema).

Encara que aquesta breu introducció a una GUI és suficient per al que farem en endavant, el lector interessat pot explorar les diferents opcions de cada menú per descobrir per si mateix altres possibilitats que li ofereix el sistema operatiu.

1.7.2 Instruccions en mode text

La GUI proporciona una imatge gràfica de l'estructuració del disc fix en directoris i fitxers molt útil per comprendre la filosofia general del sistema operatiu. És convenient, no obstant això, tenir una idea de com es treballa en mode text, la qual cosa s'entendrà de seguida executant instruccions que reprodueixin algunes de les tasques que es poden realitzar de manera gràfica.

Per obrir una consola o terminal (en Unix/Linux s'anomena també una *shell*) clicarem sobre la icona que normalment representa un monitor o una petxina (si aquesta no està en la barra de menús s'haurà de buscar en el menú d'aplicacions). A la part superior esquerra de la consola apareixerà una línia de text —el *prompt* o *indicador*— que, depenent de com estigui configurat el sistema, pot indicar el nom de l'usuari, una identificació de l'ordinador, el nom del directori on s'està treballant, etc. A continuació del *prompt* hi ha el *cursor*, que és un quadradet (o rectangle) que assenyala la posició de la finestra on començarà a aparèixer el text que escriurem des del teclat. El *prompt* que trobareu, juntament amb el cursor, pot ser d'aquest estil:

```
[/home/usuari1] $ ■
```

L'aparició del *prompt* indica que l'ordinador està en disposició de rebre instruccions del sistema operatiu. Per executar-ne una hem d'escriure-la i prémer la tecla de retorn (⏎). Així, si escrivim:

[13] Si aquesta icona no està en l'escriptori s'haurà d'executar la instrucció `mount` en mode text.

```
pwd
```

(abreviació de *print working directory*) i premem [↩], ens apareixerà una indicació de quin és el *directori de treball*, és a dir, el directori on s'està treballant (també se'n diu directori *actiu* o directori en què *s'està*). Si aquest és el directori de l'usuari apareixerà:

```
/home/usuari1
```

Per treure una llista dels fitxers ubicats en aquest directori i els subdirectoris que hi pengen podem executar la instrucció `ls` (abreviació de *list*). Els ítems d'aquesta llista es corresponen amb les icones que apareixen en clicar sobre la icona de la carpeta de l'usuari. Si volem que aquesta llista inclogui més informació de cada fitxer o subdirectori podem afegir l'opció `-l` a la instrucció `ls`:

```
ls -l
```

Si, per exemple, hi ha dos fitxers anomenats `fitxer1` i `fitxer2` i dos subdirectoris anomenats `cartes` i `informes` apareixerà una cosa semblant a això:

```
total 4
drwx------    3 usuari1  usuari1      1360 nov 27 20:04 cartes
-rw-r--r--    1 usuari1  usuari1        48 nov 27 20:04 fitxer1
-rw-r--r--    1 usuari1  usuari1        62 nov 27 20:04 fitxer2
drwxr-xr-x    2 usuari1  usuari1      1360 nov 27 20:04 informes
```

és a dir, se'ns indica quin és el nombre total d'ítems —fitxers o subdirectoris— i s'escriu una línia per a cadascun d'aquests ítems. El primer caràcter d'aquesta línia és una `d` si l'ítem és un subdirectori i un guió quan es tracta d'un fitxer normal. A continuació s'indica el grau de protecció que té cada ítem (les lletres `r`,`w` i `x` indiquen permisos de lectura, escriptura i execució per part de diferents categories d'usuari), l'*inode* (un nombre que té un significat massa tècnic per ser descrit aquí) seguit del nom del propietari de l'ítem i el del grup d'usuaris al qual pertany, la memòria que ocupa en el disc mesurada en bytes, la data i l'hora de la seva última modificació i el seu nom.

Per crear un subdirectori del directori actiu s'utilitza la instrucció `mkdir` (*make directory*), equivalent a l'opció *Crear nova carpeta* del menú *Editar* de la GUI:

```
mkdir nomsubdir
```

on `nomsubdir` és el nom que vulguem donar al nou subdirectori. Per *passar* a aquest, és a dir, per fer que el nou subdirectori sigui l'actiu, usarem la instrucció `cd` *(change directory)*:

```
cd nomsubdir
```

En executar-la el *prompt* canviarà per indicar que ens trobem en el subdirectori `nomsubdir` del directori de l'usuari; per exemple:

```
[/home/usuari1/nomsubdir] $ ■
```

Aquest subdirectori encara no tindrà cap ítem, com es pot comprovar executant la instrucció `ls`. El directori del qual penja, és a dir, l'anterior directori de treball, es pot representar abreujadament mitjançant dos punts seguits. Així, per tornar a aquell directori des del `nomsubdir` podem fer:

```
cd ..
```

La instrucció `cd` permet passar directament a qualsevol directori identificat mitjançant la seva designació completa; per exemple, la instrucció

```
cd /home/usuari2
```

ens passarà al directori de l'usuari2 (si no està protegit). Després podrem executar la instrucció

```
cd /home/usuari1
```

per tornar al nostre directori.

Per referir-nos a un subdirectori o fitxer del directori actiu no cal incloure la indicació d'aquest últim. Així, tot i que es podria passar des del directori de l'usuari1 al subdirectori `nomsubdir` amb la instrucció:

```
cd /home/usuari1/nomsubdir
```

és més pràctic fer-ho de la manera que hem vist abans (`cd nomsubdir`).

Per duplicar un fitxer, és a dir, per crear-ne un de nou que contingui la mateixa informació que l'original s'utilitza la instrucció `cp` (*copy*). Per exemple, si el fitxer original s'anomena `fitxer1` i volem que el nou s'anomeni `fitxer2`, ambdós ubicats al directori actiu, farem:

```
cp fitxer1 fitxer2
```

Si la ubicació d'un d'aquests fitxers no és el directori actiu haurem d'indicar-ho; per exemple:

```
cp fitxer1 nomsubdir/fitxer2
```

copiarà el fitxer `fitxer1` del directori actiu en un nou fitxer del subdirectori `nomsubdir` anomenat `fitxer2`. Si s'omet el nom del nou fitxer s'agafarà el mateix nom que tenia l'original; així, la instrucció

```
cp fitxer1 nomsubdir
```

farà una còpia del fitxer `fitxer1` al subdirectori `nomsubdir` amb el mateix nom. Per fer aquesta mateixa operació des del directori `nomsubdir` (després d'haver fet `cd nomsubdir`) podem usar la instrucció:

```
cp ../fitxer1 .
```

on el punt que apareix al final identifica el directori actiu.

La instrucció `rm` (*remove*) permet esborrar fitxers, la qual cosa equival a llençar-los a la paperera des de la GUI i buidar-la (o seleccionar l'opció *Esborrar* després de clicar-hi amb el botó dret). Així,

```
rm informe
```

esborra del directori actiu un fitxer anomenat `informe`.

Per moure un fitxer d'un directori a un altre, la qual cosa equival a copiar-lo i esborrar després l'original, usarem la instrucció `mv` (*move*). Per exemple, executar la instrucció

```
mv document1 nomsubdir
```

equival a executar les dues instruccions

```
cp document1 nomsubdir
rm document1
```

La instrucció `mv` també permet canviar el nom d'un fitxer; per exemple, perquè el fitxer `fitxer1` passi a anomenar-se `fitxer2` farem:

```
mv fitxer1 fitxer2
```

En moltes instruccions, com ara `cp`, `mv` i `rm`, es pot utilitzar un asterisc per representar qualsevol seqüència de caràcters. Per exemple,

```
cp fitxer* nomsubdir
```

copia tots els fitxers el nom dels quals comenci per `fitxer` al subdirectori `nomsubdir`, i

```
rm *
```

esborra tots els fitxers del directori actiu.

Per eliminar un subdirectori s'han d'esborrar primer tots els fitxers que conté i, després, executar la instrucció `rmdir` (*remove directory*):

```
rmdir nomsubdir
```

Si volem veure el contingut d'un fitxer anomenat `informe` i no necessitem modificar-lo podem utilitzar la instrucció `more`:

```
more informe
```

Si el contingut del fitxer no cap a la consola, l'escriptura s'atura quan la consola està plena, i s'ha de prémer la tecla d'espaiat per prosseguir la visualització. Per tornar a la pàgina anterior s'ha de prémer la tecla [b]. Es pot abandonar la visualització abans d'arribar al final del fitxer prement la tecla [q].

Una instrucció molt útil és `man` (de *manual*), que ens mostra la part del manual de referència del sistema que explica el funcionament d'una instrucció i les opcions que admet. Així, la instrucció:

```
man cp
```

presenta informació sobre la instrucció `cp`. Es pot controlar la visualització de la manera descrita per a la instrucció `more`.

Per als que coneguin el sistema operatiu DOS pot ser interessant saber que algunes de les seves instruccions funcionen en GNU/Linux afegint davant la lletra `m`. Per exemple, la instrucció:

```
mformat a:
```

formata o *inicialitza* un disquet, prèviament introduït en la disquetera, la qual cosa esborra tot el que pogués contenir i el prepara per gravar-hi informació. La instrucció

```
mcopy *.f a:
```

copia tots els fitxers el nom dels quals acabi en `.f` al disquet. Aquesta instrucció munta i desmunta el disquet cada vegada que s'executa; en conseqüència, aquest no haurà d'estar muntat i es podrà expulsar prement el botó d'expulsió directament, com es fa al sistema Windows. Sempre s'ha d'esperar, però, que s'apagui el llumet que hi ha al costat del botó.

Per acabar, un truc que estalvia temps i errades: el sistema memoritza les instruccions que executem en mode text, i podem recuperar-les amb les tecles [↑] i [↓] per no haver de tornar a teclejar-les. Una instrucció recuperada es por modificar abans de tornar-la a executar (es poden emprar les tecles [→] i [←] per desplaçar el cursor sobre el text).

Exercici 1.11
Escriviu les instruccions de GNU/Linux necessàries per crear un subdirectori de l'actiu amb nom `programes`, passar al nou subdirectori, moure-hi els fitxers del directori anterior el nom dels quals acabi en `.f` i veure la relació d'ítems que conté el nou subdirectori.
Solució:

```
mkdir programes
cd programes
mv ../*.f .
ls
```

2 Programació en llenguatge Fortran 77

2.1 Llenguatges de programació

Suposem que tenim un ordinador equipat amb un sistema operatiu i una sèrie d'aplicacions i volem utilitzar-lo per resoldre un problema numèric que no es pot abordar amb aquest programari. Una solució és fer una aplicació específica per al tipus de problema que tenim plantejat, per a la qual cosa haurem de disposar d'un procediment per transmetre-li les instruccions necessàries. Això es pot fer mitjançant un *llenguatge de programació*, que és un conjunt de paraules (un *vocabulari*) i de convenis per combinar-les (una *sintaxi*) que permeten codificar *instruccions* en una forma intel·ligible per als ordinadors. Encara que tenen moltes similituds amb els llenguatges ordinaris, el seu vocabulari és molt més restringit, la qual cosa els fa més fàcils d'aprendre, si bé la seva sintaxi és molt més estricta: petits errors poden fer que l'ordinador no entengui el que li volem transmetre. Les paraules del vocabulari d'un llenguatge de programació se solen anomenar *paraules clau*. Un *programa* és una successió d'instruccions, expressades mitjançant un llenguatge de programació, que fa una tasca determinada.

Cada tipus de processador té un llenguatge de programació propi que es diu *llenguatge màquina*. Aquest no fa servir les lletres i els altres símbols que s'utilitzen per transcriure els llenguatges parlats; les instruccions del llenguatge màquina es codifiquen fent servir només els dígits 0 i 1, els mateixos que s'utilitzen per emmagatzemar informació en la memòria. Tot i que aquest llenguatge és l'únic que entén directament el processador, no s'utilitza perquè té certs inconvenients:

- és molt difícil d'entendre per a nosaltres, que estem acostumats a utilitzar una col·lecció de caràcters molt més àmplia,
- diferents tipus de processadors tenen llenguatges màquina diferents, la qual cosa fa que un programa creat per a un processador no es pugui executar en un d'un altre tipus,
- els programes són llargs i complicats, perquè les instruccions són molt bàsiques (copiar una dada d'una posició de la memòria RAM a un registre, sumar les xifres guardades en dos registres i posar el resultat en un altre, etc.).

El primer inconvenient es pot pal·liar fent servir el *llenguatge d'assemblador*, que utilitza lletres, dígits decimals (0 a 9) i altres caràcters per representar les instruccions del llenguatge màquina. Encara que això el fa més fàcil d'entendre per a una persona, es tracta d'una transcripció molt directa del llenguatge màquina que segueix essent dependent del tipus de processador i dóna lloc a programes llargs i complicats. La traducció al llenguatge màquina es fa mitjançant un tipus d'aplicació que s'anomena *assemblador*. Aquests dos llenguatges —màquina i assemblador— són molt "propers" al processador i se'n diuen *de baix nivell*.

Els llenguatges *d'alt nivell* són molt més fàcils d'entendre per a les persones, ja que utilitzen paraules mnemotècniques per construir instruccions. A més, aquestes poden ser força *potents* —en el sentit que una sola instrucció pot equivaler a diverses instruccions en llenguatge màquina—, la qual cosa permet reduir molt la llargada dels programes. D'altra banda, la forma de les instruccions

d'un llenguatge d'alt nivell no depèn del tipus de processador, de manera que el programa no s'haurà de modificar si canviem de tipus de màquina. Perquè pugui funcionar un programa escrit en un llenguatge d'alt nivell s'ha de traduir al llenguatge màquina de l'ordinador on el volem executar, la qual cosa es fa mitjançant certes aplicacions que seran considerades en la secció següent.

Hi ha bastants llenguatges d'alt nivell, concebuts per a diferents àmbits d'aplicació. El pioner fou el *Fortran* (The IBM Mathematical FORmula TRANslating System, 1954-1957), un llenguatge orientat al càlcul matemàtic molt utilitzat en l'àmbit científic.[1] Més tard es desenvolupà el *Cobol* (COmmon Business Oriented Language, 1959), un llenguatge dissenyat per tractar problemes de gestió (manipular bases de dades, portar la comptabilitat d'empreses, etc.). El *Basic* (Beginner's All Purpose Symbolic Instruction Code) fou introduït el 1964 com a versió simplificada del Fortran per a principiants. El *Pascal* (1970) fou concebut per aprendre a programar. El *C* (1973) és un llenguatge d'aplicació general molt estès en l'àmbit de la informàtica professional. El *C++* és una evolució del llenguatge C apareguda cap a 1981. El *Java*, per la seva banda, es dissenyà a principis de la dècada dels noranta. Tots ells tenen molts elements comuns i el coneixement d'un llenguatge facilita força l'aprenentatge d'altres.

Hi ha d'altres llenguatges informàtics de propòsit menys general, com ara el *PostScript* (1982), pensat per codificar la informació que s'envia a les impressores per imprimir text i gràfics; el *TeX* (1986), que s'utilitza per escriure textos amb fórmules matemàtiques; el *html* (HyperText Markup Language, 1990) i el *JavaScript* (1995), que s'utilitzen per confeccionar pàgines web, etc.

2.2 Intèrprets i compiladors

Disposem de dos tipus d'aplicacions per traduir un programa escrit en un llenguatge d'alt nivell (el *codi font*) guardat en un fitxer (el *fitxer font*) al llenguatge màquina (el *codi objecte*) d'un ordinador concret: els intèrprets i els compiladors.

Un *intèrpret* és una aplicació que llegeix una a una les instruccions d'un programa escrit en un llenguatge d'alt nivell; després de llegir cada instrucció, la tradueix a una o diverses instruccions de llenguatge màquina i fa que s'executin. Les instruccions traduïdes no es guarden a cap fitxer. Si l'intèrpret detecta algun error en una instrucció ho indica mostrant un missatge i s'atura perquè puguem corregir-la. Una vegada corregida prossegueix la interpretació. Això facilita força la *depuració* —localització i correcció d'errors— del programa. La interpretació, però, és un sistema poc eficient per executar programes llargs i/o repetitius, ja que, d'una banda, s'ha de traduir individualment cada instrucció que s'executa i, de l'altra, aquesta traducció s'ha de repetir cada vegada que es vol utilitzar el programa. Els programes escrits en Basic s'executen sovint mitjançant interprets. Els clients o visualitzadors de pàgines web, també anomenats *navegadors d'Internet* (Explorer, Firefox, Mozilla, Netscape, Opera, Safari...) són intèrprets dels llenguatges que s'utilitzen per confeccionar pàgines web (html, JavaScript, etc.). Quan obrim una pàgina des d'un navegador, aquest va llegint, a través de la xarxa Internet, les instruccions d'un fitxer (o més d'un) ubicat en un servidor de pàgines web, que és un altre ordinador connectat a la xarxa. Aquestes instruccions contenen informació sobre com s'ha de presentar la pàgina en el nostre ordinador: grandària i tipus de lletra, colors, imatges, so, etc. A mesura que les llegeix, el navegador les interpreta i, si les entén, les executa (per això, de vegades, la pàgina apareix en el monitor de manera gradual). De vegades el navegador no entén alguna instrucció i presenta un missatge d'error.

Un *compilador* és una aplicació que tradueix a llenguatge màquina un programa sencer escrit en un llenguatge d'alt nivell i guarda la versió traduïda en un nou fitxer, que anomenarem *fitxer objecte*. La CPU d'un ordinador només pot fer operacions molt elementals (sumar, multiplicar, etc.) i, com ja hem indicat, els llenguatges d'alt nivell disposen d'instruccions potents que permeten efectuar operacions molt més complexes, com ara calcular funcions trigonomètriques, logaritmes, arrels quadrades, etc. Això fa que s'hagi d'afegir al fitxer objecte el codi objecte de *subprogrames* adients per poder fer aquestes operacions. Aquests subprogrames se subministren amb el compilador agrupats en *llibreries*, i s'incorporen al fitxer objecte en una fase posterior a la compilació pròpiament dita, anomenada *muntatge* (o *linkatge*) del programa. Aquesta segona etapa produeix un *fitxer executable*, que és, de fet, una aplicació. Per executar el programa només caldrà teclejar el nom del

[1] El cap de l'equip que va desenvolupar la primera versió del Fortran fou John Warner Backus (3/12/1924 - 17/3/2007).

fitxer executable i prémer la tecla de retorn, si estem treballant en mode text, o clicar sobre la seva icona en una GUI.[2] Alguns compiladors poden encadenar la compilació i el muntatge automàticament, tot deixant només el fitxer executable. La paraula *compilació* s'utilitza en aquest cas per referir-se al conjunt de les dues etapes.

L'execució de programes compilats és més ràpida que la dels programes interpretats, no tan sols perquè no cal traduir les instruccions cada vegada, sinó també perquè la compilació sol optimitzar certes parts del programa per tal d'accelerar la seva execució. La correcció d'errors és, però, una mica més lenta, ja que cada error, per petit que sigui, ens obliga a modificar el fitxer font i compilar-lo de nou. Els programes escrits en Fortran se solen traduir a llenguatge màquina mitjançant compiladors.

Els errors es poden detectar en qualsevol etapa del procés. Així, una paraula clau escrita malament genera, normalment, un error *de compilació* o *de sintaxi*, però si la paraula errònia és el nom d'una crida a un subprograma complementari l'error es detectarà a la fase *de muntatge*. A més, pot ser que el programa no faci la tasca prevista perquè estigui dissenyat malament, tot i no detectar-se cap error en la compilació ni en el muntatge. Llavors l'execució donarà un resultat incorrecte (si el dóna). Per poder detectar aquests errors *d'execució* convé provar el programa per a diferents casos i, si és possible, amb dades que condueixin a resultats previsibles.

2.3 Primeres nocions del llenguatge Fortran

La resta d'aquest llarg capítol es dedicarà a introduir les instruccions més bàsiques de la versió 77 del llenguatge Fortran.[3] Encara que existeixen versions posteriors que eliminen algunes limitacions del Fortran 77 (la manca de recursivitat, l'assignació estàtica de la memòria, etc.), aquesta versió és força senzilla i, per tant, adequada per al nivell introductori d'aquest text. Una vegada assolit aquest nivell és fàcil passar, de manera autodidacta, a altres llenguatges o a altres versions del Fortran.

Certs aspectes de la discussió subsegüent —com ara la forma concreta en què s'escriuen les dades en format lliure— depenen del compilador utilitzat. En aquests casos l'exposició es referirà al compilador lliure *g77*, que es distribueix juntament amb el sistema operatiu GNU/Linux, el MacOs X i altres sistemes derivats de l'Unix.

2.3.1 Un exemple preliminar

Encara que, potser, la manera més elegant de presentar un llenguatge de programació consisteix a descriure la seva sintaxi i els diferents tipus d'instruccions i, a continuació, il·lustrar cadascuna d'aquestes instruccions mitjançant exemples, en certs casos és més pedagògic començar amb exemples molt senzills que permetin captar l'objectiu del que es vol introduir i procedir, després, a descriure-ho de manera més formal. Així, iniciarem aquesta introducció al Fortran amb l'anàlisi d'un programa per calcular la mitjana aritmètica de dos nombres. El primer que farem és obrir un nou fitxer de text i escriure-hi les instruccions o *sentències* que s'indiquen a continuació (observeu que els espais en blanc s'han indicat mitjançant el símbol ␣, i que a cada línia hi ha sis espais en blanc abans de la primera lletra):[4]

```
␣␣␣␣␣␣read␣(5,*)␣a,␣b
␣␣␣␣␣␣c␣=␣(a+b)/2
␣␣␣␣␣␣write␣(6,*)␣"La␣mitjana␣es",␣c
␣␣␣␣␣␣end
```

Guardarem aquest fitxer en el disc dur amb un nom amb extensió `.f` (per indicar que conté un programa Fortran), per exemple, `mitjana.f`. Abans de compilar-lo analitzarem, un mica per sobre, el que farà cada instrucció. La primera línia conté una instrucció (`read (5,*) ...`) que permetrà introduir en l'ordinador des del teclat (dispositiu o *unitat* que s'identifica en Fortran amb el número 5) les dades necessàries per calcular la mitjana, és a dir, els nombres que volem amitjanar. Aquests dos nombres es guardaran en sengles zones de la memòria RAM —*variables*— que hem anomenat `a` i `b`. La segona línia agafa les dades guardades en `a` i `b`, en calcula la mitjana i la guarda en la variable

[2] Certs executables requereixen sempre una consola.

[3] En el moment d'escriure aquest text la darrera versió del Fortran data del 2003.

[4] En els programes Fortran no utilitzarem accents, atès que en algunes instal·lacions no s'interpreten correctament.

`c`, i la tercera (`write (6,*) ...`) escriu en el monitor (unitat 6) el valor guardat en aquesta variable. L'última instrucció (`end`) indica al compilador que el programa s'ha acabat.

Per compilar el programa font contingut en el fitxer `mitjana.f` amb el compilador g77 i, si no es detecten errors, muntar-lo i generar el corresponent fitxer executable (que anomenarem `mitjana`) s'ha de teclejar en una consola la instrucció següent:

```
g77 -o mitjana mitjana.f
```

i prémer la tecla [↩].[5] Si el compilador detecta algun error no es generarà el fitxer executable i caldrà editar el fitxer font per corregir l'error, guardar-lo i compilar-lo de nou. Quan la compilació no dóna errors es crea el fitxer executable en el mateix directori del fitxer font. Per executar-lo escriurem el seu nom precedit de la seva ubicació:

```
./mitjana
```

i premerem la tecla [↩]. La indicació que el fitxer executable està ubicat en el directori actiu (`./`) garanteix que s'executi el programa que hem fet, ja que podria haver-hi una instrucció del sistema operatiu amb el mateix nom que el nostre programa.

En executar-se el programa, aquest quedarà aturat esperant que teclegem els dos nombres que volem amitjanar separats per un espai en blanc o per una coma. Una vegada els haguem teclejat hem de prémer la tecla [↩], la qual cosa farà que aquestes dades es guardin en les variables `a` i `b` (instrucció `read (5,*) a, b`) i continuï l'execució del programa. Si, per exemple, introduïm les dades 4 i 6.5 (en Fortran s'utilitza el punt decimal en comptes de la coma per a dades numèriques), la instrucció `write (6,*) "La mitjana es", c` farà apareixer en la consola el missatge:[6]

```
La mitjana es  5.25
```

i, en la línia següent, apareixerà el *prompt* indicatiu que l'ordinador queda en disposició de rebre noves instruccions del sistema.

Observem que quan comença l'execució del programa no apareix cap indicació del que s'ha de fer. Nosaltres sabem que hem d'introduir dos nombres perquè coneixem el codi font, però si passem l'executable a una altra persona no sabrà que ha de fer. És convenient dissenyar els programes de manera que puguin ser utilitzats per una persona —l'*usuari*— distinta de la que ha fet el programa —el *programador*—, que pot no conèixer el codi font ni, fins i tot, el llenguatge utilitzat. Això exigeix que el programa informi clarament a l'usuari del que ha de fer en cada moment. A part de facilitar la utilització del programa per altres persones, això pot ser útil, fins i tot, per al mateix programador: si necessita emprar un programa que ha fet fa temps podrà fer-ho sense haver de mirar el fitxer font per recordar quines dades s'han d'introduir. Així, podríem incloure, abans de la primera instrucció, les tres instruccions `write` següents:

```
␣␣␣␣␣␣write␣(6,*)␣"Teclegeu␣els␣nombres␣que␣voleu␣amitjanar,"
␣␣␣␣␣␣write␣(6,*)␣"separats␣per␣un␣espai␣en␣blanc␣o␣per␣una␣coma,"
␣␣␣␣␣␣write␣(6,*)␣"i␣premeu␣Retorn"
```

les quals faran aparèixer en la consola els missatges següents:

```
Teclegeu els nombres que voleu amitjanar,
separats per un espai en blanc o per una coma,
i premeu Retorn
```

abans que s'aturi l'execució del programa a l'espera de les dades per a la instrucció `read`.

2.3.2 Tipus d'instruccions

En el programa de la pàgina 19 totes les instruccions, excepte l'última, codifiquen alguna acció que es farà durant l'execució: guardar en la memòria RAM dades introduïdes a través del teclat, operar amb aquestes dades i guardar en memòria el resultat de l'operació, fer aparèixer algun missatge al monitor, etc. Direm, per això, que són instruccions *executables*.

En general, els programes constaran de tres parts:

[5] Si s'utilitza el compilador *gfortran*, que també és lliure, s'ha de canviar la paraula `g77` per `gfortran`.

[6] El nombre de decimals amb què s'escriu el resultat pot variar d'un compilador a un altre.

- en primer lloc, unes instruccions *de declaració*, que defineixen certes característiques de les dades que utilitzarà el programa,
- a continuació les instruccions *executables*, que fan que l'ordinador efectuï certes operacions (no necessàriament de tipus matemàtic),
- finalment, la instrucció `end` indica que el programa s'ha acabat.

Pot no haver-hi instruccions de declaració, com succeeix en l'exemple anterior. Si n'hi ha, han d'estar abans que la primera instrucció executable.

Hi ha un quart tipus d'instrucció que, sense ser executable, pot anar a la mateixa zona del programa que aquestes. Es tracta de la instrucció `format`, que permet indicar com s'han d'introduir a l'ordinador les dades que requereix un programa i com s'han de presentar els resultats que aquest produeix. Si no s'inclou cap instrucció `format` les lectures i escriptures de dades s'efectuaran d'una manera predefinida pel compilador. Es tracta, doncs, d'una instrucció prescindible, la principal aplicació de la qual és millorar l'estètica de la presentació de resultats. En aquest text prescindirem d'aquesta instrucció fins a la secció 3.4, on se n'inclou una breu descripció.

Les instruccions executables es poden classificar en tres grups:

- les *d'entrada* o *lectura* de dades i les *de sortida* o *escriptura* de resultats permeten que l'usuari intercanviï informació amb l'ordinador,
- les *d'assignació* permeten efectuar un càlcul i guardar-ne el resultat en la memòria RAM,
- les *de control* permeten alterar l'ordre seqüencial d'execució de les instruccions del programa.

Al programa de l'apartat anterior hi ha instruccions de sortida (`write`), d'entrada (`read`) i d'assignació (`c = (a+b)/2`).

A part de les instruccions pròpiament dites, es poden incloure *línies de comentaris* abans o després de qualsevol instrucció d'un programa Fortran. Aquestes no són traduïdes pel compilador (les omet), i la seva finalitat és donar al programador, sigui el mateix autor o un revisor del programa, indicacions que el facin més fàcil d'entendre: assenyalar diferents etapes del programa, explicar el significat de certes variables, etc. Qualsevol que escrigui un programa una mica complicat i vulgui modificar-lo un temps després comprendrà de seguida la utilitat d'aquest tipus d'indicacions.

2.3.3 Sintaxi

Les instruccions del llenguatge Fortran 77 s'han d'ajustar a un format força rígid. Aquest format ve determinat per les regles següents:

- Si hi ha una `c` (o `C`) a la *posició* (o *columna*) 1, el compilador entén que es tracta d'una línia de comentari i, per tant, no la tradueix.
- Només s'utilitzen les posicions 1 a 72 de cada línia.[7]
- Les instruccions pròpiament dites no poden començar abans de la posició 7 (encara que poden començar més a la dreta d'aquesta).
- Les posicions 1 a 5 d'una instrucció executable o d'una instrucció `format` es poden emprar per posar-hi una *etiqueta*. Aquesta és un nombre enter i positiu que pot anar de l'1 al 99999 i serveix per poder fer referència a la instrucció des d'una altra sentència del programa. Les etiquetes no tenen per què estar ordenades. Les etiquetes són prescindibles si no es fan servir instruccions format, i en aquest text només s'utilitzaran a les seccions 3.4, 3.7 i 3.8.

[7] La limitació a 72 posicions té una explicació històrica: quan els programes s'escrivien sobre fitxes perforades (vegeu la nota al peu de la pàgina 9), les posicions 73 a 80 es reservaven per numerar-les (la utilitat d'aquesta pràctica no se solia valorar prou fins que no queia a terra el paquet de fitxes!). D'altra banda, la raó que es parli sovint de "columnes" en comptes de "posicions" rau, probablement, en el fet que la informació de cada fitxa estava disposada en forma de taula, i cada caràcter ocupava una de les 80 columnes d'aquesta taula (el caràcter apareixia escrit a la part superior de la columna i les perforacions que el codificaven, a sota).

- Si una instrucció és tan llarga que no cap en una sola línia es posa un caràcter qualsevol (que no sigui un zero o un blanc) a la posició 6 (*posició* o *columna de continuació*) de la línia següent i la resta de la instrucció a partir de la posició 7. Si la instrucció completa encara no hi cap, es poden emprar més línies de continuació.

- El compilador prescindeix dels espais en blanc que troba entre las posicions 7 i 72 excepte dintre d'un text entre cometes; per tant, podem intercalar els espais que vulguem per tal de facilitar la lectura del programa.

- El compilador no distingeix entre lletres majúscules o minúscules excepte dintre d'un text entre cometes.

Així, l'última instrucció executable del programa de la pàgina 19 es podria haver escrit amb menys espais en blanc i amb la paraula *write* en majúscules:

```
WRITE(6,*)"La mitjana es",c
```

però si traguéssim els espais del text entre cometes i l'escrivíssim amb majúscules:

```
write (6,*) "LAMITJANAES", c
```

es veurien enganxades i amb majúscules les paraules al missatge de presentació del resultat.

A continuació presentem una versió d'aquell programa que inclou tres línies de comentaris i agrupa les tres instruccions `write` inicials en una de sola:

```
c La linia que ve a continuacio facilita la identificacio de la posicio
c de cada caracter en la versio impresa del codi font:
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      write (6,*) "Teclegeu els nombres que voleu amitjanar, separats pe
     $r un espai en blanc o per una coma, i premeu Retorn"
      read (5,*) a, b
      c = (a+b)/2
      write (6,*) "La mitjana es", c
      end
```

Observeu que la primera instrucció executable del programa, que ocupa dues línies, escriurà correctament el missatge que demana les dades: després de la posició 72 de la primera línia s'escriurà el caràcter de la posició 7 de la segona (una 'r'). L'única funció del caràcter que ocupa la posició 6 (`$`) és indicar al compilador que la línia on es troba conté la continuació de la instrucció de la línia anterior. Les dues primeres línies de comentaris informen (al programador) sobre la finalitat de la tercera: numerar les posicions de cada línia per tal de poder identificar amb facilitat les posicions 7 i 72.

La forma general d'una instrucció d'assignació (sense etiqueta) és:

```
variable = expressió
```

on el significat precís de la paraula "variable" i la sintaxi de les expressions es veuran en els propers apartats. Qualsevol altra instrucció comença amb una paraula clau del llenguatge Fortran que permet al compilador identificar el tipus d'instrucció:

```
paraula_clau  resta_de_la_instrucció
```

Per exemple, les paraules clau `read` i `write` permeten reconèixer les instruccions de lectura i escriptura de dades, respectivament. S'acostuma a utilitzar aquestes paraules clau per referir-se a cada tipus d'instrucció: la instrucció `read`, la instrucció `write`, etc.

Fora dels textos entre cometes i del caràcter de continuació de la posició 6, només s'utilitzen les lletres de l'alfabet anglès, dígits numèrics (sense subíndex ni superíndex), espais en blanc i els caràcters especials + − * / = . , ( ) ' : ".

2.3.4 Variables i constants

Quan un programa s'executa fa servir un cert nombre de dades que, mentre dura l'execució, resideixen a la memòria RAM de l'ordinador. Per exemple, el programa de l'apartat anterior ha de guardar a la memòria les dues dades que introdueix l'usuari en executar-se la instrucció `read`: la primera es guarda en una zona de la memòria RAM identificada mitjançant la lletra `a`, i la segona dada es guarda en la zona `b`. Aquestes zones de memòria reben el nom de *variables*, i tenen un paper semblant al de les memòries d'una calculadora de butxaca: podem recuperar la informació que contenen i modificar-la. La principal diferència és que el nombre de memòries d'una calculadora sol ser força reduït, en tant que el nombre de variables que podem utilitzar en un programa és enorme (tot i que estarà limitat per la memòria RAM de l'ordinador).

Cada variable s'identifica en el programa mitjançant un nom que, en Fortran 77, ha de tenir d'1 a 6 caràcters alfanumèrics (lletres de l'alfabet anglès i/o dígits numèrics) dels quals el primer ha de ser forçosament una lletra.[8] Per exemple, `radi` o `index5` són noms vàlids, però `$super` o `3D` no ho són. El programa que hem fet utilitza tres variables, els noms de les quals són `a`, `b` i `c`. En alguns llenguatges s'ha d'especificar al principi de cada programa (en la zona de les instruccions de declaració) totes les variables que s'utilitzaran, però en Fortran s'entén que qualsevol seqüència de caràcters (no declarada) que comenci per una lletra, no sigui una paraula clau i no estigui entre cometes és una variable. Això explica que no hagin calgut instruccions de declaració en el nostre programa.

De vegades interessa guardar a la memòria RAM dades que no s'han de modificar durant l'execució del programa. Els valors d'aquestes dades s'assignen en començar l'execució i romanen fixos durant aquesta, per la qual cosa els anomenem *constants*. En un programa Fortran es poden fer servir constants no declarades escrivint el seu valor on es necessiti; per exemple, al nostre programa hem utilitzat la constant 2 per calcular la mitjana en la instrucció `c = (a+b)/2` . També es poden usar *constants figuratives*, que són constants a les quals s'ha assignat un nom i un valor mitjançant una instrucció de declaració `parameter`:

```
parameter (const1 = valor1, const2 = valor2 ...)
```

Per exemple, si un programa ha d'emprar diverses vegades els nombres π i e és més pràctic declarar-los com a constants figuratives mitjançant la instrucció

```
parameter (pi = 3.14159265, e = 2.71828183)
```

i escriure els noms de les constants `pi` o `e` en comptes dels seus valors cada vegada que es necessitin; per exemple:

```
circum = 2*pi*r
area = pi*r*r
x = 1/e
```

Observem que, si en una instrucció `parameter` declarem més d'una constant, aquestes han d'anar separades per comes. Els noms de les constants figuratives s'han d'ajustar a les mateixes regles que els de les variables.

És evident que, en l'exemple anterior, es podrien haver utilitzat variables en lloc de les constants figuratives `pi` i `e`:

```
pi = 3.14159265
e = 2.71828183
circum = 2*pi*r
area = pi*r*r
x = 1/e
```

o, també, constants no figuratives:

```
circum = 2*3.14159265*r
area = 3.14159265*r*r
x = 1/2.71828183
```

[8] Alguns compiladors n'admeten més de 6.

però la utilització de les constants figuratives presenta certs avantatges:

- a diferència de les variables, les constants figuratives no es poden modificar en la resta del programa (el compilador donaria un missatge d'error); això permet evitar modificacions no desitjades d'aquestes dades com a conseqüència, per exemple, d'errors del programador.
- si s'ha de fer servir una mateixa constant en diversos punts del programa, és més còmode i menys subjecte a errors definir-la una sola vegada com a figurativa que tornar a escriure el seu valor cada vegada;
- si haguéssim de modificar el valor d'una constant serà més ràpid i segur fer-ho una sola vegada al principi del programa que haver de corregir-la a cada lloc on s'utilitza, la qual cosa comportaria més risc d'errors en teclejar-la i la possibilitat d'oblidar-nos de modificar alguna de les aparicions de la constant.

2.3.5 Tipus de dades

Les variables i les constants poden ser de diferents tipus, segons el tipus de dades que continguin: enteres, reals, reals de precisió doble, literals, complexes, lògiques, etc.

Enteres

Les variables o constants *enteres* només poden emmagatzemar nombres enters, és a dir, sense decimals. El seu valor pot ser, doncs, 10000, –156, o bé 0, però no 1225,1. En el cas de les constants el valor s'escriu sense punt decimal; així, la constant `2` que apareix a la instrucció `c = (a+b)/2` del programa de la pàgina 22 és entera, però no ho seria si s'hagués escrit en la forma `2.0`

Totes les variables i constants figuratives que comencen per alguna de les lletres (majúscules o minúscules) 'i', 'j', 'k', 'l', 'm', 'n' són enteres si no s'indica que el seu tipus és un altre en alguna instrucció de declaració; per exemple, `k10`, `longit` o `j`. Si volem que ho sigui una variable (o constant figurativa) que comenci per una altra lletra haurem d'indicar-ho, la qual cosa es pot fer mitjançant la instrucció de declaració explícita `integer`:

```
integer var1 , var2 ...
```

Per exemple, la primera de les instruccions següents

```
integer z, b5, total
parameter (total = 100, minim = -50)
```

declara com a enteres les variables `z` i `b5` i la constant `total`, i la segona assigna els valors 100 i –50 a les constants enteres `total` i `minim`, respectivament.

Actualment la major part dels ordinadors fan servir 4 bytes de memòria (4×8 = 32 bits) per guardar cada variable o constant entera; això permet codificar 2^{32} nombres enters diferents. Com que un bit es reserva per al signe, podrem codificar des del $-2^{31} = -2\ 147\ 483\ 648$ fins al $2^{31} - 1 = 2\ 147\ 483\ 647$ (el zero es codifica amb el bit del signe en positiu).

Reals

Les variables o constants *reals* accepten nombres amb decimals. El seu valor pot ser 0,5; –3,14; 1,0; etc. (tingueu en compte que aquests valors s'escriuran: `0.5`; `-3.14`; `1.0`; etc.). En el cas de les constants, el valor s'ha d'escriure amb punt decimal (fins i tot quan no té decimals); per tant, si al programa de la pàgina 22 haguéssim escrit la instrucció que calcula la mitjana així

```
c = (a+b)/2.
```

la constant `2.` estaria codificada com a nombre real, no com a enter (vegeu la secció 1.3). La part entera es pot ometre si és zero; així, `-.05` és equivalent a `-0.05`. Totes les variables i constants figuratives no declarades explícitament que comencen per alguna lletra (majúscula o minúscula) diferent de 'i', 'j', 'k', 'l', 'm' o 'n' són reals; per exemple, `a`, `area` o `radi`. Les variables `a`, `b` i `c` de l'exemple de la pàgina 22 són, doncs, reals. Si volem que ho sigui una variable (o constant figurativa) que comenci per 'i', 'j', 'k', 'l', 'm' o 'n' podem indicar-ho mitjançant la instrucció de declaració explícita `real` :

```
real var1, var2 ...
```

Per exemple, la primera de les instruccions

```
real j, n, massa, m5
parameter (pi = 3.14159265, m5 = -.5)
```

declara com a reals les variables `j`, `n` i `massa` i la constant `m5`, i la segona assigna els valors 3,14159265 i –0,5 a les constants reals `pi` i `m5`, respectivament.

El valor de les constants reals es pot expressar en notació científica; per exemple, la instrucció

```
parameter (h = 6.6260693e-34)
```

assigna el valor 6,6260693 $\times 10^{-34}$ a la constant `h` i la instrucció

```
eps = 1e-8
```

guarda el valor 10^{-8} a la variable `eps` (observeu que el punt decimal es pot ometre en la notació científica quan no hi ha decimals). L'exponent de 10 ha de ser enter (no podem posar, per exemple, `5.5e1.5`).

Atès que els enters són un subconjunt dels reals pot semblar innecessari utilitzar variables o constants enteres, però això no sempre és així. Una raó és que les dades enteres es codifiquen de manera exacta i, en canvi, les reals no (en general). Així, el programa següent:

```
x = 4.8
write (6,*) x
end
```

escriu:

```
 4.80000019
```

A la major part dels ordinadors, les variables reals ocupen 4 bytes, dels quals un s'utilitza per a l'exponent de 10. D'aquesta manera es poden codificar xifres des de l'ordre de $\pm 10^{-42}$ fins a $\pm 10^{38}$ amb una precisió de 7 xifres o dígits significatius (vegeu l'apèndix C.1). La declaració de variables reals de 4 bytes es pot fer també substituint la paraula clau `real` per `real*4`, la qual cosa garanteix que les variables declarades ocuparan 4 bytes independentment del compilador que s'utilitzi.

Reals de precisió doble

Si es vol que els càlculs amb nombres reals tinguin més precisió que la que proporcionen les variables i constants reals, o si s'han d'utilitzar valors que estan fora de l'interval que admeten aquelles dades, es poden fer servir dades *reals de precisió doble*. Aquestes ocupen 8 bytes, tenen una precisió de 16 xifres significatives i abasten des de l'ordre de $\pm 10^{-321}$ fins a $\pm 10^{308}$ (aquestes xifres poden variar depenent de la instal·lació).

Per indicar el valor d'una constant real de precisió doble s'ha d'utilitzar la notació científica amb una 'd' en comptes de la 'e': `15.33456723451201d12`, `5d-6`, `0d0`, etc. Les variables i constants figuratives de precisió doble s'han de declarar sempre, la qual cosa es pot fer mitjançant la instrucció de declaració explícita `double precision`:

```
double precision var1, var2 ...
```

També es pot utilitzar la instrucció equivalent `real*8`. Per exemple, les instruccions

```
double precision a1,a2,a3,num
real*8 long1, z, g
parameter (num = 5d-50, g = 2.0023193043737d0)
```

declaren les variables `a1`, `a2`, `a3`, `long1` i `z` i les constants `num` i `g` com a reals de precisió doble, tot assignant els valors de les dues últimes. S'ha d'especificar la notació 'dn' (fins i tot si n val 0) en el valor assignat a una constant figurativa per tal que aquest valor es guardi amb precisió doble. Així, el programa

```
double precision pi
parameter (pi = 3.1415926535897932)
write (6,*) pi
end
```

escriu `3.14159274`; en canvi, aquest altre:

```
double precision pi
parameter (pi = 3.14159265358979320d0)
write (6,*) pi
end
```

escriu `3.14159265` i, si fem servir una instrucció format perquè ho presenti amb més decimals (vegeu la secció 3.4), escriurà `3.14159265358979310`.

Literals

Les variables i les constants *literals* permeten emmagatzemar *cadenes* de caràcters; per exemple, el nom o el cognom d'una persona, el nom d'un fitxer, la successió de caràcters "april 14th, 1931" (els espais en blanc i els signes de puntuació compten com a caràcters), etc. Les variables literals s'han de declarar sempre. Això es pot fer amb la instrucció `character*n`:

```
character*n var1 , var2 ...
```

on `n` indica el nombre màxim de caràcters que poden guardar les variables declarades. Aquests caràcters poden ser qualssevol dels de la codificació ASCII (vegeu la secció 1.3). Per exemple,

```
character*12 nom, cognom, fitxer, digits
```

declara com a literals les variables `nom`, `cognom`, `fitxer` i `digits` tot admetent cadenes de 12 caràcters com a màxim. El valor d'una constant literal s'ha de posar entre apòstrofs o cometes:

```
nom = 'Maria'
cognom = "Guanyabens"
fitxer = "ex1.f"
digits = "1234567890"
```

S'ha de tenir en compte que la darrera d'aquestes instruccions assigna els 10 caràcters '1', '2', ..., '9', '0' a la variable `digits`, la qual cosa no és el mateix que assignar el valor 1234567890 a una variable entera. En aquest últim cas es codifica en binari el nombre enter 1.234.567.890, i en el primer es codifica cada xifra independentment mitjaçant el sistema ASCII, de manera que no es podrà utilitzar el contingut de la variable `digits` en operacions matemàtiques.

Per incloure un apòstrof com a caràcter en una constant literal es poden usar les cometes o duplicar l'apòstrof, i viceversa; per exemple: `"L'arrel"`, `'L''arrel'`, `'un preu "tope" car'`, `"un preu ""tope"" car"`, etc. La major part dels compiladors fan servir 1 byte per guardar cada caràcter d'una dada literal.

Altres tipus de dades

El Fortran també permet usar nombres *complexos* (de precisió normal o doble) i dades de tipus *lògic*, que només poden tenir dos valors: `.true.` (cert) i `.false.` (fals). Alguns compiladors admeten també dades reals i complexes de precisió quàdrupla. Cap d'aquests tipus serà utilitzat en aquest text.

Declaracions implícites

El Fortran permet fer *declaracions implícites* de tipus de dades, en les quals s'indica la primera lletra de les variables o constants a declarar, en comptes d'una relació concreta de noms. Per exemple, la instrucció:

```
implicit double precision (a, b, c)
```

declara com a reals de precisió doble les variables que comencen per 'a', 'b' o 'c' (`a`, `b1`, `corba`, etc.). D'una manera més general, la instrucció:

```
implicit double precision (inicial1-inicial2, inicial3-inicial4 ...)
```

declara com a reals de precisió doble totes les variables que comencin per les lletres de l'abecedari compreses entre la *inicial1* i la *inicial2*, o entre la *inicial3* i la *inicial4*, etc. Així, si hem utilitzat variables reals no declarades en un programa i veiem que cal augmentar la precisió dels càlculs que fa, podem afegir al principi del programa la instrucció

```
implicit double precision (a-h, o-z)
```

per convertir-les a precisió doble. En efecte, aquesta declaració afectarà totes les variables (o constants) que comencin per alguna lletra compresa entre la 'a' i la 'h' o entre la 'o' i la 'z'; és a dir, les que no comencen per 'i', 'j', 'k', 'l', 'm' ni 'n' (que en absència de declaracions serien enteres). Com que, avui dia, els ordinadors solen tenir memòria RAM de sobres per a la major part dels programes, molts programadors acostumen a posar aquesta instrucció o l'equivalent

```
implicit real*8 (a-h, o-z)
```

al principi de cada programa Fortran per tal de minimitzar els errors numèrics implícits en les operacions amb nombres reals.

Encara que l'aplicació més comuna de les instruccions de tipus `implicit` és la declaració de variables de precisió doble, es pot aplicar a qualsevol altre tipus de variable; per exemple,

```
implicit real (a-z)
```

declararia com a reals normals o *de precisió senzilla* totes les variables del programa.

Les instruccions de declaració implícita s'han de posar abans que qualsevol altra instrucció de declaració. Les declaracions fetes mitjançant instruccions de *declaració explícita* de tipus de dades (`integer`, `real`, `character`, etc.) prevalen sobre las especificades mitjançant instruccions de declaració implícita. Així, les instruccions

```
implicit double precision (a-h, o-z)
integer ordre, digit
```

declaren com a variables de precisió doble totes les que comencin per una lletra compresa entre 'a' i 'h' o bé entre 'o' i 'z' excepte `ordre` i `digit`, que seran enteres.

Exercici 2.1
Indiqueu la quantitat de memòria RAM, expressada en bytes, que ocuparan, conjuntament, les variables que apareixen en el fragment de programa següent (incloses les declarades explícitament i les que apareixen en la instrucció `read`):

```
implicit double precision (a-h, o-z)
real arrel, long
integer quanti
read (5,*) arrel, long, quanti, x, y, z, l
```

Solució: 40 bytes.

2.4 Instruccions d'entrada i de sortida

Hem vist que les instruccions d'entrada i de sortida de dades permeten que l'usuari intercanviï informació amb l'ordinador. S'anomenen també instruccions de lectura i escriptura. Aquests termes s'han d'interpretar des del punt de vista de l'ordinador, no del de l'usuari: una instrucció de lectura fa que la CPU llegeixi informació mitjançant algun dispositiu d'entrada, i una d'escriptura fa que l'ordinador escrigui alguna cosa en un dispositiu de sortida.

La instrucció de lectura del Fortran s'identifica mitjançant la paraula clau `read`, i la seva forma més comuna és:

```
read (5,*) var1, var2, ..., varn
```

on el nombre que va al començament del parèntesi indica el perifèric o *unitat* pel qual s'efectuarà l'entrada de les dades (el 5 fa referència al teclat),[9] l'asterisc indica que la lectura es farà en *format lliure*, és a dir, d'una manera predefinida (dades separades per comes o espais en blanc)[10] i *var1*, *var2*, ..., *varn* són noms de n variables qualssevol del programa. Aquesta llista no pot contenir constants, ni numèriques ni literals (textos entre cometes).

Quan arribi a aquesta instrucció l'ordinador aturarà l'execució del programa esperant que l'usuari teclegi n dades separades per comes o espais en blanc i premi la tecla de retorn. En fer-ho la primera dada es guardarà a la primera variable, la segona dada es guardarà a la segona variable, etc. També es pot escriure la primera dada i prémer la tecla de retorn, escriure la segona i tornar a prémer Retorn, etc. Com a regla general, cada dada introduïda s'ha de correspondre amb el tipus de la variable on es guardarà, encara que l'ordinador pot fer certes conversions. Per exemple, si donem un valor enter (és a dir, sense punt decimal) a una variable real la conversió a aquest tipus és automàtica, però si donem un valor amb punt decimal com a dada per a una variable entera l'execució del programa s'aturarà (*avortarà*), i donarà un missatge d'error.[11] Les dades per a variables literals (cadenes de caràcters) es poden teclejar entre cometes o apòstrofs (els apòstrofs o cometes no es guarden).

Al programa de la pàgina 22 hi ha una instrucció de lectura:

```
read (5,*) a, b
```

En arribar a aquesta instrucció el programa s'atura fins que l'usuari teclegi dos nombres (que poden tenir decimals) i premi la tecla de retorn. A partir d'aquest moment, la variable real `a` prendrà el primer valor i la `b`, el segon.

La instrucció d'escriptura del Fortran s'identifica mitjançant la paraula clau `write` i la seva forma més comuna és:

```
write (6,*) varoco1, varoco2, ..., varocon
```

on el 6 indica que la sortida dels resultats es farà pel monitor, l'asterisc indica que es farà en format lliure (valors separats per espais en blanc),[12] i *varoco1*, *varoco2*, ..., *varocon* són n variables i/o constants. En executar-se aquesta instrucció s'escriuran en el monitor els valors emmagatzemats en cada una d'aquestes constants o variables.

El programa de la pàgina 22 conté dues instruccions `write`; la primera té només una constant literal (que ocupa dues línies), i la segona inclou una constant literal i una variable real:

```
write (6,*) "La mitjana es ", c
```

El text contingut en la constant literal s'escriurà tal qual: `La mitjana es:`. A continuació es deixaran un o més espais en blanc i s'escriurà el valor que conté la variable `c` en el moment d'executar-se la instrucció d'escriptura. Just abans s'haurà executat la instrucció `c = (a+b)/2`, de manera que el valor que s'escriurà serà la mitjana de les dades entrades per l'usuari.

Cada vegada que s'executa una instrucció `write` es comença a escriure en una nova línia. Així, les instruccions

```
write (6,*) a
write (6,*) b
```

escriuran els valors que tenen les variables `a` i `b` en línies diferents, mentre que la instrucció

```
write (6,*) a, b
```

escriurà els dos valors a la mateixa línia. Anàlogament, la instrucció

```
read (5,*) a, b
```

[9] Aquest nombre pot fer referència a un perifèric o a un fitxer concret emmagatzemat en un perifèric, tant en les instruccions de lectura com en les d'escriptura (vegeu la secció 2.12).

[10] En la secció 3.4 s'indiquen altres maneres de fer la lectura.

[11] Pot ser que altres compiladors acceptin la dada real, i la converteixin a entera.

[12] La manera concreta d'expressar cada valor (part entera, decimals...) i el nombre d'espais que els separen depèn del compilador.

permet llegir dues dades de cop (tot prement una sola vegada la tecla de retorn) i amb les instruccions

```
read (5,*) a
read (5,*) b
```

hem de prémer Retorn després de teclejar cada dada.

Observem que una instrucció `read` modifica els valors de les variables que hi apareixen, cosa que no succeeix amb una instrucció `write`.

Ja hem esmentat la conveniència de dissenyar els programes de manera que puguin ser executats per usuaris que no coneguin el codi font. D'acord amb aquesta pràctica, no convé posar una instrucció `read` sense incloure abans una instrucció `write` que informi l'usuari sobre les dades que ha d'entrar. A més, si el programa ha de llegir el valor d'una magnitud física és important que indiqui a l'usuari les unitats del valor que s'ha d'entrar, les quals dependran de com ho hagi previst el programador. Suposem, per exemple, que un programa calcula el volum que ocupa un gas ideal a partir del nombre de mols n, de la pressió P i de la temperatura T fent servir l'equació $PV = nRT$, i aquestes dades són les que es demanen a l'usuari. En demanar la temperatura, el programa ha d'informar l'usuari que es tracta de la temperatura absoluta expressada en kelvins; així, es podria posar:

```
write (6,*) "Introdueix la temperatura expressada en kelvins"
read (5,*) tempab
```

Anàlogament, s'han d'indicar les unitats de les altres dades de manera que siguin coherents amb el valor assignat a la constant R, i, s'han d'indicar també les unitats del resultat; per exemple, si aquest està contingut en la variable `vol` i les seves unitats són litres:

```
write (6,*) "Volum =", vol, "l"
```

En les instruccions `read` i `write` es pot substituir el nombre 5 o 6 que identifica la unitat estàndard de lectura o escriptura per un asterisc; per exemple:

```
write (*,*) "Introdueix la temperatura expressada en kelvins"
read (*,*) tempab
```

De vegades s'utilitzen les paraules angleses *input* i *output* per referir-se, respectivament, al conjunt de dades que ha de llegir un programa i als resultats que escriu.

2.5 Instruccions d'assignació. Operadors aritmètics

Ja hem vist (apartat 2.3.3) que una instrucció d'assignació (sense etiqueta) té la forma general:

variable = expressió

L'expressió és una successió de variables i/o constants separades per operadors. Es poden utilitzar també parèntesis. Si aquestes dades són numèriques els operadors que es poden usar són els *aritmètics*: `+` per a la suma, `-` per a la resta, `*` per a la multiplicació, `/` per a la divisió i `**` per a la potenciació. En executar-se una instrucció d'assignació s'avalua el resultat de l'expressió i, després, es guarda el valor obtingut en la variable que figura a l'esquerra del signe '='. També es diu que s'ha assignat aquell valor a aquesta variable. En el moment de fer-se l'assignació la variable perd el valor que tenia fins aleshores i agafa el nou. En el programa de la pàgina 22 n'hem vist un exemple:

```
c = (a+b)/2
```

Observem que el significat que té en Fortran el signe '=' és diferent del que té en una equació matemàtica. En Fortran no representa una igualtat entre dues expressions, sinó *una transferència d'informació*, ja que *trasllada a la variable de l'esquerra el resultat de l'expressió que hi ha a la dreta.*[13] Convé tenir present que l'avaluació de l'expressió és prèvia al trasllat del resultat a la variable. Així, les instruccions Fortran

[13] Més endavant introduirem un *metallenguatge* en el qual utilitzarem una fletxa que apunta cap a l'esquerra (←) —més representativa que el signe '='— per identificar les assignacions.

```
i = 3
i = i+1
```

tenen sentit (tot i que la segona seria absurda com a equació matemàtica): la primera assigna el valor 3 a la variable `i` i la segona suma 1 al valor d'aquesta variable (3) i guarda el resultat de la suma (4) a la mateixa variable `i`; és a dir, aquesta perd el valor que tenia i adquireix el valor 4.

Exercici 2.2
Feu un programa que llegeixi una temperatura en graus Celsius, la converteixi en temperatura absoluta expressada en kelvins i l'escrigui. Recordeu que

$$T/\mathrm{K} = t/^\circ\mathrm{C} + 273,15$$

Suggeriment: Feu una primera versió amb dues variables —una per a la temperatura en graus Celsius i una altra per a l'absoluta— i una segona versió que utilitzi una sola variable.

L'assignació d'un valor enter a una variable real comporta la recodificació del valor enter a real; és a dir, la seva *conversió* en una dada real. Així, el programa

```
m = 5
x = m
write (6,*) x
end
```

escriurà `5.` (amb el punt decimal). Anàlogament, quan s'assigna un valor enter o real a una variable real de precisió doble aquell es recodifica com a real de precisió doble. Com que la codificació d'un real de precisió senzilla només garanteix unes 7 xifres significatives correctes, la seva conversió a real de precisió doble mantindrà aquest grau de precisió, el qual està molt per sota de les 16 xifres significatives garantides en una dada real de 8 bytes. Així, en executar el programa

```
double precision x
x = 4.8
write (6,*) x
end
```

es troba el resultat `4.80000019`; en canvi, si assignem a la variable `x` la mateixa constant codificada amb precisió doble:

```
x = 4.8d0
```

s'escriu exactament la xifra que hem assignat: `4.8`

L'assignació d'un valor real (de precisió senzilla o doble) a una variable entera comporta la pèrdua dels decimals que pogués tenir aquell valor (el *truncament* dels decimals), de manera que el programa

```
x = 4.8
m = x
write (6,*) m
end
```

escriurà un `4.` Observem que no es tracta d'un arrodoniment: tot i que el nombre real 4,8 està més proper al 5 que al 4, el truncament dels decimals el transforma en un 4.

Una operació aritmètica entre dos nombres del mateix tipus dóna un valor d'aquest mateix tipus. En particular, la divisió entre enters dóna un enter, que és igual a la part entera del quocient; és a dir, aquesta operació comporta el truncament dels decimals que pogués tenir el quocient real. Per exemple, el programa

```
i = 3
k = i/4
write (6,*) k
end
```

escriurà un zero, ja que la part entera de 3/4 = 0,75 és 0. Declarar la variable `k` com a real no solucionaria el problema, ja que els decimals es perden en avaluar-se l'expressió `i/4`, la qual cosa es fa abans d'assignar-se el resultat a la variable `k`. Si haguéssim afegit un punt a la constant del denominador (`k = i/4.`) la divisió donaria el valor real 0,75, ja que quan un dels operands d'una operació aritmètica és enter i l'altre és real es converteix el primer a real abans d'efectuar-se l'operació. Això, però, tampoc soluciona el problema si no s'acompanya de la declaració de la variable `k` com a real per tal d'evitar el truncament dels decimals en fer-se l'assignació.

Anàlogament, una operació entre un enter o un real de precisió senzilla i un de precisió doble comporta la conversió prèvia del primer a real de precisió doble.

Exercici 2.3
Indiqueu el resultat que escriurà el programa següent:

```
a = 1.
b = 2.
n = a/b
i = a
j = b
c = i/j
d = a/b
write (6,*) n, c, d
end
```

Solució: 0 0. 0.5

Exercici 2.4
Feu un programa que llegeixi dos valors reals per a sengles variables `a` i `b` i permuti els valors d'ambdues variables, de manera que, en finalitzar l'execució, la variable `b` tingui el valor que s'havia llegit per a la variable `a` i aquesta tingui el valor prèviament llegit per a la variable `b`. Incloeu una instrucció `write (6,*) a,b` a continuació de la lectura de les dades i una altra idèntica després de fer l'intercanvi per tal de comprovar el funcionament correcte del programa. *Suggeriment*: Un algorisme senzill per resoldre el problema es pot il·lustrar mitjançant el símil següent: les variables `a` i `b` s'assimilarien a dos gots, A i B, que, en comptes de tenir emmagatzemats valors numèrics, contindrien volums diferents d'aigua. Per intercanviar el contingut dels gots podem utilitzar un tercer got auxiliar C: en un primer pas aboquem l'aigua del got A al C, després transvasem el contingut del B al A i finalment passem el del C al B.

2.5.1 Ordre d'avaluació d'operacions aritmètiques

Considerem l'expressió aritmètica `a+b/c**2`; amb el que hem vist fins ara no podem saber quina de les operacions matemàtiques següents representa:

$$a + \frac{b}{c^2}, \qquad \frac{a+b}{c^2}, \qquad a + \left(\frac{b}{c}\right)^2 \quad \text{o} \quad \left(\frac{a+b}{c}\right)^2$$

Per resoldre aquest tipus d'ambigüitats el llenguatge Fortran estableix el següent ordre de prioritat decreixent per a les operacions aritmètiques:

1. `**`
2. `*` i `/`
3. `+` i `-`

Per exemple, en l'expressió abans considerada s'avaluarà la potenciació en primer lloc, després el quocient i finalment la suma; l'expressió Fortran representa, doncs, la primera de les operacions matemàtiques indicades.

Si una expressió aritmètica conté més d'un operador del mateix nivell de prioritat, aquests s'avaluen d'esquerra a dreta excepte en el cas de les potenciacions consecutives, en què és de dreta a esquerra. Així, `a+b-d/x*y` representa $a+b-\frac{d}{x}y$, no $a+b-\frac{d}{xy}$, i `a**b**c` representa $a^{(b^c)}$, no $(a^b)^c$.

Podem usar parèntesis per forçar un ordre d'avaluació diferent de l'establert per les regles anteriors; així, per programar l'expressió $\frac{a+b}{c+d}$ no podem escriure `a+b/c+d`, que representaria $a+\frac{b}{c}+d$, i haurem d'utilitzar dos parèntesis: `(a+b)/(c+d)`. D'altra banda, es poden usar més parèntesis dels indispensables per tal de facilitar la interpretació d'una expressió; per exemple, `a**(b**c)`, o `a+b-(d/x)*y`.

Exercici 2.5
Indiqueu les expressions matemàtiques que representen les expressions Fortran següents:
a) `a+b/c**2*d`
b) `x/y/z+1`
c) `a*(1+x)**2/3`
d) `a/(b*c**(d/e))`
Solució: a) $a+\frac{b}{c^2}d$; b) $\frac{x}{yz}+1$; c) $a\frac{(1+x)^2}{3}$; d) $\frac{a}{bc^{d/e}}$.

Exercici 2.6
Feu un programa que demani a l'usuari el nombre de bitllets de 10, 20 i 50 euros que té a la cartera i calculi i escrigui el valor total d'aquests bitllets.

Exercici 2.7
Feu un programa que escrigui la part entera del quocient i també el residu de la divisió entre dos nombres enters positius introduïts com a dades. *Indicació*: El residu (r) s'obté restant del dividend (dd) el producte del divisor (ds) per la part entera del quocient (qe): $r = dd - ds \times qe$; per exemple, la part entera del quocient entre 17 i 3 és 5 i el residu és 2.

Exercici 2.8
Feu un programa que calculi i escrigui el volum en litres que ocupa un gas ideal ($PV = nRT$) a partir de les dades següents (que haurà de demanar a l'usuari): nombre de mols n, temperatura absoluta T (en kelvins) i pressió P en bars (1 bar = 10^5 pascal = 0,986923 atm). Utilitzeu una instrucció `parameter` per definir el valor de la constant dels gasos $R = 0{,}08314472$ bar l K^{-1} mol^{-1} i incloeu una línia de comentari per indicar al programador les unitats d'aquest valor. Penseu a indicar a l'usuari les unitats de les dades i les del resultat. *Comprovació*: Per a 1 mol de gas a una temperatura de 273,15 K i 1 bar de pressió s'obté un volum de 22,7110 l, i per a 1,5 mol de gas en les mateixes condicions s'obté un volum de 34,0665 l.

Exercici 2.9
a) Feu un programa que calculi la pressió P (en bars) d'un gas ideal a partir del volum V que ocupa (en litres), la temperatura T (en Kelvin) i el nombre de mols n.
b) L'equació d'estat d'un gas segons el model de van der Waals és:

$$\left(P + \frac{an^2}{V^2}\right)(V - nb) = nRT$$

on R és la constant dels gasos i els paràmetres a i b depenen del gas considerat. Feu un programa que llegeixi els paràmetres a (en l^2 bar mol^{-2}) i b (en l mol^{-1}) d'un gas determinat i utilitzi l'equació de van der Waals per calcular la pressió P (en bars) de n mols de gas que ocupen un volum V (en litres) a una temperatura T (en K). Tingueu en compte que $R = 0{,}08314472$ bar l K^{-1} mol^{-1}.

Comprovació: Per a 1,5 mols de N_2 (a = 1,370 l^2 bar mol^{-2}, b = 0,0387 l mol^{-1}) que ocupen 34,0665 l a una temperatura de 273,15 K s'obté una pressió de 0,999050 bar. Observeu que la pressió obtinguda fent servir l'equació de van der Waals s'assembla molt al valor corresponent a un gas ideal en les mateixes condicions (apartat *a*), que és d'un bar. En canvi, si reduïm el volum a la desena part (3,40665 l), s'obté una pressió d'uns 9,90773 bar, amb una desviació relativa major respecte del valor ideal (10 bar).

Exercici 2.10
Modifiqueu qualsevol dels programes dels apartats *a)* i *b)* de l'exercici anterior perquè demani la temperatura en graus Celsius (T/K = t/°C + 273,15). *Comprovació*: Feu servir les dades de l'exercici anterior, tot expressant la temperatura en graus Celsius (0°C).

Exercici 2.11
Feu un programa que demani a l'usuari el radi r d'una esfera expressat en cm i calculi el volum que quedarà lliure quan la fiquem en la capsa cúbica més petita en la qual càpiga. *Comprovació*: Per a un radi de 10 cm el volum lliure és de 3811,21 cm^3.

2.6 Funcions incorporades

A més de les operacions aritmètiques considerades en la secció anterior, el Fortran permet avaluar moltes operacions matemàtiques mitjançant *funcions incorporades* o *intrínseques* que depenen d'una o més variables o constants. Aquestes dades s'anomenen els *arguments* de la funció i s'escriuen entre parèntesis a continuació del nom d'aquesta (separats per comes si n'hi ha més d'un). Per exemple, per avaluar el sinus d'un angle d'1,5 radians podem utilitzar la instrucció:

```
x = sin(1.5)
```

i la instrucció:

```
x = exp(p)
```

permet calcular l'exponencial del valor contingut en la variable `p` (e^p). Cada funció té un nombre i tipus d'arguments que s'han de respectar. Així, les funcions `sin` i `exp` tenen un sol argument real; en canvi, la funció `mod`, que proporciona el residu de la divisió entre dos enters, té dos arguments enters: el primer és el dividend i el segon, el divisor. Per exemple, el programa

```
k = 14
l = mod(k,5)
write (6,*) l
end
```

escriurà un `4.`

Els arguments poden ser expressions, que seran avaluades abans de calcular-se la funció; per exemple, `mod((i+j)/2,m*n)`. D'altra banda, les funcions poden formar part d'expressions aritmètiques i, en aquest cas, s'avaluaran abans d'efectuar-se les operacions aritmètiques. Així, l'expressió Fortran `a+exp(x+y)/sin(x)**2` representa l'operació $a + \frac{e^{(x+y)}}{\sin^2 x}$.

Algunes funcions incorporades usuals són:
`abs(x)`: valor absolut de l'argument real `x`.
`sqrt(x)`: arrel quadrada de l'argument real `x`.
`exp(x)`: exponencial de l'argument real `x`.
`log(x)`: logaritme neperià de l'argument real `x`.
`log10(x)`: logaritme decimal de l'argument real `x`.
`sin(x)`: sinus de l'angle real `x` expressat en radians.
`cos(x)`: cosinus de l'angle real `x` expressat en radians.
`tan(x)`: tangent de l'angle real `x` expressat en radians.

`asin(x)`: arc sinus expressat en radians del nombre real `x`.
`acos(x)`: arc cosinus expressat en radians del nombre real `x`.
`atan(x)`: arc tangent expressat en radians del nombre real `x`.
`mod(n,m)`: residu (enter) de la divisió de l'enter `n` entre el `m`.
`max(x1,x2 ...)`: nombre més gran de la llista d'arguments d'un mateix tipus `x1`, `x2` ...
`min(x1,x2 ...)`: nombre més petit de la llista d'arguments d'un mateix tipus `x1`, `x2` ...

En la relació anterior s'entén que els reals poden ser de precisió senzilla o doble. En alguns compiladors de Fortran s'ha d'afegir una 'd' al davant del nom de la funció si l'argument és real de precisió doble (`dabs(x)`, `dsqrt(x)`, etc.).

Si un programa utilitza funcions incorporades, la informació necessària per poder-les avaluar s'afegeix al programa compilat en el moment de fer-se el muntatge.

Exercici 2.12
Feu un programa que demani a l'usuari la concentració de H^+ en una dissolució aquosa expressada en mol/dm^3, calculi el pH de la dissolució (pH = –log[H^+]) i el pOH (pOH = 14–pH) i els escrigui. *Comprovació*: Per a [H^+] = 0,01 mol/dm^3 s'ha d'obtenir pH = 2 i pOH = 12.

Exercici 2.13
La mecànica quàntica descriu l'estat de menor energia d'un àtom d'hidrogen mitjançant l'orbital ϕ_{1s}, que és la funció de la distància electró-nucli (r) següent:

$$\phi_{1s}(r) = \frac{1}{\sqrt{\pi a_0^3}} e^{-r/a_0}$$

on $a_0 = 5,29177 \times 10^{-11}$ m. El quadrat d'aquesta funció representa la probabilitat per unitat de volum (o densitat de probabilitat) de trobar l'electró en un punt qualsevol de l'espai situat a una distància r del nucli. Escriviu un programa que calculi el valor de la funció $(\phi_{1s})^2$, expressat en el SI, per a una distància electró-nucli introduïda per l'usuari en nm (1 nm = 10^{-9} m). El programa haurà de convertir aquest valor a metres. En previsió que el resultat pugui ser un valor molt petit (és a dir, amb una potència de 10 molt negativa) convé que utilitzeu variables de precisió doble (π = 3,1415926535897932...). Executeu el programa per a diferents valors de r i comproveu que la densitat de probabilitat disminueix ràpidament en augmentar la distància de l'electró al nucli. *Comprovació* : Densitats de probabilitat en m^{-3}: $4{,}90514\times10^{28}$ per a r = 0,1 nm; $8{,}28110\times10^{13}$ per a r = 1 nm; $1{,}55760\times10^{-134}$ per a r = 10 nm.

Exercici 2.14
A 800°C la velocitat de la reacció $2N_2O(g) \rightarrow 2N_2(g) + O_2(g)$ segueix una equació cinètica de primer ordre:

$$v = -\frac{1}{2}\frac{d[N_2O]}{dt} = k[N_2O]$$

amb $k = 1{,}7$ s^{-1}.
a) Integreu l'equació anterior (a mà) per obtenir l'expressió que dóna la concentració de N_2O en funció del temps.
b) Feu un programa que calculi la concentració de N_2O en mol/l a un temps donat en ms per a una concentració inicial determinada (aquesta concentració inicial i el temps es demanaran a l'usuari com a dades). *Comprovació*: a) $[N_2O] = [N_2O]_0 e^{-2kt}$; b) Per a una concentració inicial de 0,20 mol/l i un temps de 100 ms s'obté una concentració de 0,142354 mol/l.

2.7 Disseny descendent d'algorismes

Un *algorisme* és una descripció no ambigua d'una seqüència de passos a seguir per resoldre —a mà o amb l'ajuda d'un ordinador— un problema. Anomenarem *acció* a cadascun d'aquests passos. Una

acció pot ser tan senzilla com escriure un resultat al monitor, o ser força més complicada, com ara calcular un producte vectorial, multiplicar dues matrius, etc.

Abans de començar a escriure un programa per resoldre un problema és convenient fer un esquema amb llapis i paper de l'algorisme que es vol utilitzar. Si el problema és complex convé descompondre'l en subproblemes més senzills, per tal que l'algorisme resultant estigui ben estructurat i sigui fàcil d'entendre i de modificar. En una primera aproximació a la resolució del problema farem un esquema molt simplificat que inclogui un nombre reduït d'accions. En una segona aproximació desglossarem les més complexes en accions més simples, i així successivament fins a arribar a una descripció prou detallada perquè es pugui transcriure fàcilment al llenguatge de programació escollit. Aquesta manera de procedir s'anomena *disseny descendent d'algorismes*.

El llenguatge que s'utilitza per fer els successius esquemes de l'algorisme no ha de ser un llenguatge de programació concret; és millor utilitzar un de més proper al nostre llenguatge parlat —*metallenguatge* o *pseudocodi*—, la qual cosa evita haver de prestar atenció als detalls sintàctics i facilita el redactat de l'esquema; ja es tindrà cura d'aquests detalls en fer la transcripció del pseudocodi al llenguatge de programació escollit. Si es decidís emprar un altre llenguatge de programació, l'esquema expressat en metallenguatge seguiria essent vàlid, i només s'hauria de refer la transcripció.

2.7.1 Estructures algorísmiques bàsiques

Qualsevol algorisme es pot descompondre mitjançant les tres *estructures algorísmiques bàsiques*: la seqüencial, l'alternativa i la repetitiva.

Estructura seqüencial

Diem que un conjunt d'accions 1, 2, ..., n, formen una *estructura seqüencial* o que s'executen de manera seqüencial si l'acció $k+1$ es comença quan s'ha acabat l'acció k.

Aquesta estructura es pot representar així:

acció 1
acció 2
. . . . . .
acció n

Estructura alternativa

En una *estructura alternativa* una acció —l'acció 1— s'executa només si es compleix una condició determinada i, si aquesta no es compleix, s'executa una altra acció —l'acció 2—.

Representarem aquesta estructura de la manera següent (les paraules subratllades són característiques de l'estructura):

si condició llavors
  acció 1
altrament
  acció 2
fi si

Observeu que hem escrit l'acció que s'executa en cada cas una mica més a la dreta que les línies que identifiquen l'estructura alternativa. Aquesta pràctica (*sagnat* de les línies) és molt útil quan es desglossa cada acció en accions més elementals, ja que permet identificar d'un cop d'ull l'inici i el final de cada bloc d'accions.

Si no s'ha d'executar cap acció quan no es compleix la condició, es pot utilitzar l'estructura simplificada següent:

si condició llavors
  acció 1
fi si

Estructura repetitiva

En una *estructura repetitiva* una acció es repeteix mentre sigui certa una condició determinada. La representarem així:

mentre condició fer
  acció
fi mentre

El primer que fa aquesta estructura és comprovar si la condició és certa. Si ho és s'executa l'acció, i es torna a avaluar la condició. Aquest procés es repeteix mentre la condició sigui certa. Cada cop que s'executa l'acció diem que s'ha fet una *iteració*. Observeu que l'acció ha de poder modificar almenys alguna de les variables que apareixen en la condició per tal que aquesta pugui deixar de complir-se i s'aturin les iteracions en algun moment.

2.7.2 Estructura repetitiva *per a*

El disseny d'un algorisme en el qual una acció s'ha de repetir un nombre predeterminat de vegades es pot simplificar utilitzant l'estructura repetitiva no bàsica (ja que es pot expressar en funció de les estructures bàsiques) *per a*:

per a i des de $n1$ fins a $n2$ amb increment $n3$ fer
  acció
fi per a

L'índex i pren el valor $n1$ la primera vegada que s'executa l'acció i s'incrementa en $n3$ a cada repetició. Les iteracions se succeeixen mentre i sigui més petit o igual que $n2$. Si no s'especifica l'increment s'entén que aquest és la unitat:

per a i des de $n1$ fins a $n2$ fer
  acció
fi per a

2.7.3 Algorismes complexos

El disseny descendent d'un algorisme complex comença amb un esquema senzill que inclou poques accions, les quals poden ser complexes; per exemple, un primer esquema d'un programa que hagi de fer 50 vegades una mateixa acció seria:

Programa exemple
  per a j des de 1 fins a 50 fer
    acció 1
  fi per a
fi programa

L'acció 1 podria ser una seqüència de dues accions més senzilles —acció 2 i acció 3—:

acció 1
  acció 2
  acció 3
fi acció

Suposem que l'acció 2 sigui prou senzilla com per no requerir més desglossament i que l'acció 3 contingui una estructura bàsica repetitiva:

acció 3
  mentre condició 1 fer
    acció 4
  fi mentre
fi acció

la qual, al seu torn, podria incloure una estructura alternativa:

```
acció 4
  si condició 2 llavors
    acció 5
  altrament
    acció 6
  fi si
fi acció
```

Si les accions 2, 5 i 6 són prou senzilles, aquest nivell de desglossament del pseudocodi serà adequat per transcriure'l al llenguatge de programació escollit. Després de fer les substitucions corresponents el programa en pseudocodi quedaria així:

```
Programa exemple
|  per a j des de 1 fins a 50 fer
|  |  acció 2
|  |  mentre condició 1 fer
|  |  |  si condició 2 llavors
|  |  |  |  acció 5
|  |  |  altrament
|  |  |  |  acció 6
|  |  |  fi si
|  |  fi mentre
|  fi per a
fi programa
```

Observem que hem afegit a l'esquema línies verticals per facilitar la identificació dels blocs d'instruccions inclosos en les diferents estructures algorísmiques.

A mesura que anem avançant en la introducció del llenguatge Fortran trobarem exemples concrets d'aquesta manera de dissenyar programes.

2.8 Estructura alternativa

Al llenguatge Fortran s'utilitzen tres instruccions per transcriure —o *implementar*— l'estructura algorísmica alternativa: `if`, `else` i `end if`. Constitueixen un tipus d'instruccions de control que anomenarem *instruccions alternatives o condicionals*. La forma general d'aquesta estructura bàsica és:

```
if (condició) then
  instruccions a executar si la condició és certa
else
  instruccions a executar si la condició és falsa
end if
```

Si la condició és certa s'executa el primer bloc d'instruccions i, si és falsa, el segon. En ambdós casos, una vegada executat el bloc corresponent continua el programa amb la instrucció següent a la `end if`. La condició és una seqüència d'expressions, generalment numèriques, relacionades mitjançant operadors de comparació i/o operadors lògics. Un *operador de comparació* permet comparar dues expressions que donin resultats del mateix tipus (enter, real, etc.); el resultat de la comparació només pot ser *cert* o *fals*.[14] La taula 2.1 indica el significat de cada un d'aquests operadors.

Exercici 2.15
Indiqueu si serà certa o falsa la condició `x**2+5*x+8.ge.exp(x)` quan la variable `x` prengui els valors:
a) 3.
b) 4.
Solució: a) Certa; b) falsa.

[14] Aquest resultat és un valor de tipus lògic (vegeu l'apartat 2.3.5).

Notació Fortran	Significat en anglès	Notació matemàtica
`.gt.`	**g**reater **t**han	$>$
`.ge.`	**g**reater or **e**qual than	$\geq$
`.lt.`	**l**esser **t**han	$<$
`.le.`	**l**esser or **e**qual than	$\leq$
`.eq.`	**eq**ual to	$=$
`.ne.`	**n**on **e**qual to	$\neq$

Taula 2.1: Significat dels operadors de comparació del llenguatge Fortran

Considerem un exemple senzill. En l'exercici 2.8 hem proposat fer un programa per determinar el volum en litres que ocupa un gas ideal donats la temperatura, la pressió i el nombre de mols, la qual cosa implica calcular nRT/P. Si l'usuari introdueix el valor zero per a la pressió la divisió entre P no podrà fer-se i, depenent del compilador, l'execució del programa s'interromprà de manera brusca amb un missatge d'error, o donarà com a resultat `INF` (infinit) per al volum. Per prevenir aquest problema convé comprovar que el valor introduït per a la pressió sigui diferent de zero abans d'operar amb ell i, en el cas contrari, no fer l'operació i escriure un missatge d'error explicatiu. De fet, tampoc no té sentit un valor negatiu per a la pressió, de manera que la comprovació del valor llegit pot fer-se de la manera següent:

```
...
write (6,*) "Entreu la pressio en bars"
read (5,*) p
if (p .le. 0.0) then
  write (6,*) "La pressio ha de ser > 0"
else
  v = n*r*t/p
  write (6,*) "Volum:",v," litres"
end if
end
```

Exercici 2.16
Modifiqueu el programa de l'exercici 2.12 per tal que, si l'usuari entra un valor nul o negatiu per a la concentració de H^+, el programa no faci cap càlcul i emeti un missatge d'error.

Exercici 2.17
Feu un programa que demani a l'usuari un nombre enter i li digui si és parell o senar. *Suggeriment*: Podeu utilitzar la funció `mod` per determinar el residu de dividir el nombre entre 2.

Exercici 2.18
Feu un programa que calculi les solucions reals d'una equació de segon grau ($ax^2 + bx + c = 0$) a partir dels coeficients (a, b i c) que determinen l'equació, els quals hauran de ser introduïts per l'usuari com a dades. Tingueu en compte que l'equació només té solucions reals si el discriminant ($b^2 - 4ac$) és ≥ 0. Si no existeixen solucions reals, el programa haurà d'indicar-ho. *Comprovació*:
i) Per a $a = 1$, $b = -4$ i $c = 3$, s'obté $x_1 = 3$ i $x_2 = 1$;
ii) per a $a = 1$, $b = 1$ i $c = 1$, el programa ha d'indicar que no hi ha solucions reals;
iii) per a $a = 2$, $b = -5$ i $c = 3$, s'obté $x_1 = 1,5$ i $x_2 = 1$ (si el programa funciona correctament amb el primer conjunt de dades i no ho fa amb aquest és probable que hagueu de repassar l'apartat 2.5.1).

Exercici 2.19
Quan el discriminant d'una equació de segon grau és negatiu, les solucions d'aquesta són nombres complexos:

$$\begin{aligned} x &= \frac{-b \pm \sqrt{b^2-4ac}}{2a} = \frac{-b \pm \sqrt{|b^2-4ac|\,(-1)}}{2a} \\ &= \frac{-b \pm \sqrt{|b^2-4ac|}\sqrt{-1}}{2a} = -\frac{b}{2a} \pm \frac{\sqrt{|b^2-4ac|}}{2a} i \end{aligned}$$

Encara que el Fortran permet treballar amb dades complexes, podem calcular solucions complexes sense utilitzar dades d'aquest tipus: haurem de calcular la part real ($-b/2a$) i la imaginària ($\sqrt{|b^2-4ac|}/2a$) de les solucions complexes i escriure-les de manera que l'usuari les vegi com a nombres complexos (part real $\pm$ part imaginària $\times\, i$). Modifiqueu el programa de l'exercici 2.18 perquè calculi també solucions complexes. *Comprovació*: Per a $a = 5$, $b = -2$ i $c = 1$, s'obté $x = 0,2 \pm 0,4\, i$.

Entre les instruccions que s'han d'executar quan la condició d'una instrucció `if` és certa o quan és falsa poden haver-hi blocs `if - else - end if` sencers. Així, en un programa que resolgui equacions de segon grau s'ha d'incloure un d'aquests blocs per distingir els casos en què hi ha solucions reals —discriminant ($b^2 - 4ac$) positiu— dels que no en tenen (exercicis 2.18 i 2.19), però també s'hauria de preveure que si l'usuari entra un coeficient $a = 0$ no es pot aplicar la fórmula:

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

En aquest cas, de fet, l'equació és de primer grau ($bx + c = 0$) i l'única solució és $x = -c/b$. Per tant, abans de resoldre l'equació s'hauria de distingir si es tracta d'una veritable equació de segon grau o si és de primer grau. Aprofitarem aquest exemple per il·lustrar el disseny descendent d'algorismes introduït en la secció 2.7.

Un primer esquema del programa podria ser el següent:

Programa Equació de segon grau
| escriure un missatge per demanar els coeficients a , b, i c
| llegir els coeficients a, b, i c
| si $a = 0$ llavors
| | resoldre l'equació de primer grau i escriure la solució
| altrament
| | resoldre l'equació de segon grau i escriure les solucions
| fi si
fi programa

Ara procedirem a detallar una mica més les accions 1 i 2:[15]

acció resoldre l'equació de primer grau i escriure la solució
| $x \leftarrow -c/b$
| escriure x
fi acció

acció resoldre l'equació de segon grau i escriure les solucions
| $d \leftarrow b^2 - 4ac$
| si $d \geq 0$ llavors
| | $x_1 \leftarrow \frac{-b-\sqrt{d}}{2a}$
| | $x_2 \leftarrow \frac{-b+\sqrt{d}}{2a}$
| | escriure x_1 i x_2
| altrament

[15] Observeu que utilitzem una fletxa que apunta cap a l'esquerra per indicar, en pseudocodi, l'assignació d'un valor a una variable i reservem el signe '=' per a les comparacions.

| | $xr \leftarrow \frac{-b}{2a}$
| | $xi \leftarrow \frac{\sqrt{|d|}}{2a}$
| | escriure $xr \pm xi * i$
| fi si
fi acció

Aquest nivell d'especificació del pseudocodi és clarament suficient perquè la transcripció al llenguatge Fortran sigui immediata (exercici 2.20).

Exercici 2.20
Modifiqueu el programa de l'exercici 2.19 (o, si no l'heu fet, el de l'exercici 2.18) perquè funcioni correctament quan el coeficient de x^2 sigui nul.

Exercici 2.21
Modifiqueu el programa de l'exercici 2.20 perquè, si els coeficients de x^2 i de x són tots dos nuls, emeti missatges adequats segons sigui nul o no el tercer coeficient.

Exercici 2.22
Feu un programa que demani a l'usuari el valor numèric de la qualificació d'un examen (q) i ens digui si correspon a un suspens ($q < 5$), aprovat ($5 \leq q < 7$), notable ($7 \leq q < 9$) o excel·lent ($q \geq 9$).

Exercici 2.23
Modifiqueu el programa de l'exercici 2.8 (pàgina 32) perquè demani a l'usuari si vol entrar la pressió en bars ($R = 0{,}08314472$ bar l K^{-1} mol^{-1}) o en atmosferes ($R = 0{,}08205746$ atm l K^{-1} mol^{-1}). *Suggeriment*: Podeu llegir una variable entera a la qual l'usuari doni el valor 1 o 2 segons vulgui utilitzar bars o atmosferes (tingueu en compte que l'usuari podria entrar un valor diferent d'aquests). *Comprovació*: Per a pressions en bars podeu utilitzar les dades indicades en l'exercici 2.8. Per a 1 mol de gas a una temperatura de 273,15 K i 1 atm de pressió s'obté un volum de 22,4140 l, i per a 1,5 mol de gas en les mateixes condicions, s'obté un volum de 33,6210 l.

Els *operadors lògics* permeten modificar o combinar *condicions elementals* —que contenen només un operador de comparació— per formar condicions més complexes. La taula 2.2 mostra els més utilitzats. L'operador `.not.` anteposat a una condició transforma aquesta en la condició contrària; per exemple, la condició `.not.(a.gt.b)` és equivalent a la condició `a.le.b`. L'operador `.and.` situat entre dos condicions dóna lloc a una condició composta que només és certa si ho són aquelles dues. Considerem, per exemple, un programa que ha de llegir una dada que representa una probabilitat; com que aquesta ha d'estar compresa en l'interval [0, 1], podem verificar-la amb el fragment de programa següent:

```
write (6,*) "Entreu la probabilitat"
read (5,*) prob
if ((prob .ge. 0.0) .and. (prob .le. 1.0)) then
  ...
else
  write (6,*) "La probabilitat ha d'estar compresa entre 0 i 1"
  ...
end if
```

L'operador `.or.` situat entre dos condicions dóna lloc a una condició composta que és certa si ho és qualsevol d'aquelles dues (tant si només ho és una d'elles com si ho són les dues). Així, l'estructura alternativa de l'exemple anterior es podria substituir per una d'equivalent fent servir la condició contrària:

```
if ((prob .lt. 0.0) .or. (prob .gt. 1.0)) then
  write (6,*) "La probabilitat ha d'estar compresa entre 0 i 1"
  ...
else
  ...
end if
```

Notació Fortran	Significat en català
`.not.`	no
`.and.`	i
`.or.`	o

Taula 2.2: Significat dels principals operadors lògics del llenguatge Fortran

A l'hora de determinar si una condició complexa és certa s'avaluen primer —si no hi ha parèntesis— les expressions aritmètiques, després s'apliquen els operadors de comparació i, finalment, els lògics. Entre aquests, `.not.` té prioritat sobre `.and.` i aquest en té sobre `.or.` Així, es pot prescindir dels parèntesis de la condició `.not.(a.gt.b)` i dels de les condicions compostes dels dos exemples anteriors, encara que és bo mantenir-los per facilitar-ne la interpretació.

Exercici 2.24
Indiqueu per a quins valors de les variables enteres `i` i `j` és certa la condició `(i.gt.0).and.(j.gt.0).or.(i.lt.0).and.(j.lt.0)`.
Solució: Per a qualsevol parella de valors no nuls del mateix signe.

Exercici 2.25
Feu un programa que demani a l'usuari el valor numèric de la qualificació d'un examen (q) i ens digui si correspon o no a un notable ($7 \leq q < 9$) utilitzant només un bloc `if - else - end if`.

Exercici 2.26
Un examen consta d'una part teòrica i una part pràctica, i la qualificació total és la mitjana aritmètica de les notes de cada part sobre 10. Per aprovar l'examen cal un mínim de 3,5 per a cada part i de 5 per al total. Feu un programa que demani a l'usuari les notes que ha tret un alumne a cada part de l'examen i indiqui si aquest està aprovat o està suspès. En el primer cas el programa ha de mostrar la qualificació numèrica total.

Exercici 2.27
Un any és bixest o de traspàs si és divisible per 4, excepte els començaments de segle (anys divisibles por 100), que només són bixests si són divisibles per 400. Feu un programa que demani a l'usuari el nombre d'un any i li digui si és o no bixest. *Comprovació*: 1900, 2006, 2007 i 2100 no ho són; 2000, 2008 i 2400 sí que ho són.

Exercici 2.28
a) Feu un programa que demani a l'usuari l'angle φ en graus sexagesimals —entre 0 i 360— que forma un vector del pla xy amb l'eix x positiu i li digui si els components cartesians del vector són del mateix signe o no sense calcular-los. Si algun dels components és nul el programa ho ha de dir. *Suggeriment*: Tingueu en compte que els vectors que tenen els components cartesians del mateix signe són els dels quadrants primer i tercer.
b) Modifiqueu el programa anterior perquè accepti angles no compresos en l'interval [0, 360], tant positius com negatius. *Suggeriment*: Si $|\varphi| > 360°$ heu de descomptar el nombre de voltes senceres.

2.8.1 Formes simplificades

En una estructura alternativa el bloc d'instruccions a executar si la condició és falsa pot estar buit, de manera que el control de l'execució ha de passar, en aquest cas, a la instrucció següent a l'`end if`:

```
if (condició) then
  instruccions a executar si la condició es certa
else
  no fer res
end if
```

Llavors es pot prescindir de la instrucció `else`, amb la qual cosa l'estructura queda de la manera següent:

```
if (condició) then
  instruccions a executar si la condició es certa
end if
```

que és la transcripció de l'estructura algorísmica alternativa simplificada:

si condició llavors
  acció
fi si

Per exemple, suposem que un programa requereix com a dada una temperatura en kelvins i volem que l'usuari pugui entrar-la tant en graus Celsius com en kelvins. Podríem llegir aquesta dada de la manera següent:

```
 character*1 u
 write (6,*) "Entreu una c minuscula si voleu introduir la temperat
$ura en graus Celsius, o qualsevol altre caracter si la voleu entra
$r en kelvins"
 read (5,*) u
 write (6,*) "Entreu la temperatura en les unitats escollides"
 read (5,*) t
 if (u .eq. "c") then
   t = t+273.15
 end if
 ...
```

En Fortran encara hi ha una forma més breu d'escriure una estructura alternativa que només es pot utilitzar quan s'ha d'executar una sola instrucció si la condició és certa i cap si és falsa:

```
if (condició) instrucció
```

Així, en l'exemple anterior podríem substituir les tres últimes instruccions per una de sola:

```
if (u .eq. "c") t = t+273.15
```

Observeu que, no s'ha de posar la paraula `then` en aquesta instrucció ni s'ha d'afegir cap instrucció `end if`; de fet, l'absència de la paraula `then` és el que indica al compilador que no es tracta d'un bloc `if - else - end if` o `if - end if`.

2.9 Instruccions de transferència de control

En un programa pot interessar que després d'executar-se una instrucció no s'executi la següent sinó una altra. Això es pot aconseguir utilitzant la instrucció de control `go to`, la qual transfereix el control de l'execució del programa a una altra instrucció identificada mitjançant una etiqueta. La utilització reiterada d'aquesta instrucció complica notablement el flux del programa, la qual cosa dificulta la localització i correcció d'errors i l'eventual introducció posterior de modificacions. D'altra banda, aquesta instrucció no és indispensable, ja que sempre es pot substituir per instruccions alternatives

i/o repetitives. Per tot això prescindirem de la instrucció `go to`, encara que, per satisfer el lector encuriosit, n'hem inclòs una breu descripció en la secció 3.7.

Una altra instrucció que, en certa manera, es pot considerar també de transferència de control és la instrucció `stop`. En executar-se s'atura l'execució del programa.

Així, si en l'exemple de la pàgina 38 el programa no s'acabés just després de la instrucció `end if` i utilitzés el volum per fer altres càlculs, s'hauria d'incloure una instrucció `stop` a continuació del missatge d'error per aturar l'execució del programa en el cas que no s'hagi pogut calcular el volum:

```
...
write (6,*) "Entreu la pressio en bars"
read (5,*) p
if (p .le. 0.0) then
  write (6,*) "La pressio ha de ser > 0"
  stop
else
  v = n*r*t/p
  write (6,*) "Volum:",v," litres"
end if
massa = v*dens
...
```

Exercici 2.29
Per tal que el programa de l'exercici 2.20 no doni error d'execució quan els coeficients de x^2 i de x siguin tots dos nuls, afegiu, a continuació de la lectura de les dades, un bloc `if - end if` adient que inclogui una instrucció `stop` (si heu fet l'exercici 2.21, pàgina 40, comprovareu que aquesta modificació és més senzilla que la que vau fer llavors).

2.10 Estructures repetitives

Una de les característiques més interessants dels ordinadors és la seva elevada velocitat de procés, la qual cosa els fa especialment útils per a feines que impliquin moltes repeticions d'un mateix conjunt d'operacions; és a dir, moltes *iteracions*. El llenguatge Fortran disposa de dos tipus de construccions per implementar algorismes iteratius:

- les instruccions `do while` i `end do` permeten transcriure l'estructura algorísmica bàsica repetitiva, i
- les instruccions `do` *amb índex* i `end do` permeten implementar l'estructura repetitiva *per a*.

La utilització d'una o altra estructura dependrà del fet que es pugui determinar o no el nombre d'iteracions abans de començar el procés iteratiu: si es coneix aquest nombre, utilitzarem el `do` amb índex i, si no, farem servir el `do while`.

Per exemple, si volem calcular la nota total d'un examen que té dues parts per als alumnes d'un grup, haurem de repetir el mateix càlcul (sumar la nota de cada part i dividir per 2) tantes vegades com alumnes tinguem. En aquest problema, doncs, coneixem d'entrada el nombre d'iteracions, i la manera més adient d'implementar-ho en Fortran serà mitjançant el `do` amb índex. Suposem, en canvi, que volem determinar una incògnita d'una equació i no podem aïllar-la; una opció és provar d'obtenir-la utilitzant algun procediment iteratiu. Si tot va bé, aquest procés convergirà cap a la solució buscada, però no sabrem d'entrada quantes iteracions caldran per obtenir-la amb el grau de precisió desitjat. Llavors haurem d'utilitzar la instrucció `do while`, que permet fer iteracions mentre es compleixi una determinada condició.

2.10.1 *Do* amb índex

Una estructura repetitiva del tipus `do` *amb índex* - `end do`, que anomenarem bloc `do - end do`, té la forma següent:[16]

```
do índex = valin,valfin,incr
  conjunt d'instruccions
end do
```

on *índex* representa una variable entera, i *valin*, *valfin* i *incr* són constants, variables o expressions numèriques amb valor enter.[17] En executar-se per primera vegada la instrucció `do` l'*índex* rep el valor *valin*; si aquest valor és $\leq valfin$ s'executa el conjunt d'instruccions que segueix fins a la instrucció `end do`, la qual retorna el control de l'execució a la instrucció `do`. Aquesta incrementa en *incr* el valor de l'*índex* i, si aquest és $\leq valfin$, es torna a executar el conjunt d'instruccions que constitueix una iteració. Aquest cicle es va repetint, incrementant-se el valor de l'*índex* en *incr* a cada iteració, fins que, en executar-se la instrucció `do`, l'*índex* prengui un valor superior a *valfin*.[18] Aleshores el control de l'execució passa a la instrucció que segueix a l'`end do`. Si no s'indica el valor d'*incr* (i no es posa la coma que el precedeix) se sobreentén que l'increment és 1. Les instruccions compreses entre el `do` i l'`end do` no poden modificar el valor de l'*índex* (si es modifica el compilador dóna un missatge d'error).

Vegem-ne un exemple. El programa següent escriu els n primers nombres naturals, essent n una dada introduïda per l'usuari:

```
write (6,*) "Escriu un nombre natural"
read (5,*) n
write (6,*) "Els ",n," primers nombres naturals son:"
do i=1,n
  write (6,*) i
end do
end
```

Si substituïm la instrucció `do` per

```
do i=n,1,-1
```

els n nombres s'escriuran en ordre decreixent, i si la substituïm per

```
do i=1,n,2
```

s'escriuran, en ordre creixent, els nombres naturals senars més petits o iguals que n. Observem que, si donem a n un valor parell, l'últim nombre que s'escriurà és $n-1$, ja que en acabar la iteració corresponent a aquest valor l'índex s'incrementarà en 2 i prendrà el valor $n+1$; com que és superior a n, el procés iteratiu s'aturarà.

Observem que hem escrit la instrucció `write` més cap a la dreta que les `do` i `end do`; aquesta pràctica de sagnar el conjunt d'instruccions que constitueix una iteració facilita la seva identificació i és molt recomanable, sobretot quan aquest conjunt és gran i quan conté altres estructures repetitives o alternatives.

Exercici 2.30
Feu un programa que demani a l'usuari un nombre natural n i escrigui els quadrats dels n primers nombres naturals.

[16] Existeix una forma alternativa del `do` amb índex que es descriu en la secció 3.8.

[17] Es pot utilitzar una variable real com a índex i dades reals per especificar els valors que aquest ha de prendre, però això pot portar problemes —especialment si l'increment no és enter i el nombre d'iteracions és elevat— com a conseqüència de l'error que comporten, en general, les operacions entre reals. Així, pot succeir que el nombre d'iteracions que s'efectuen sigui diferent del previst pel programador.

[18] Aquesta discussió només és vàlida si $incr > 0$. Si fos $incr < 0$, llavors l'índex aniria disminuint i hauria de ser $valfin \leq valin$.

Exercici 2.31
Feu un programa que demani a l'usuari un nombre natural n i escrigui els n primers termes de la successió de terme general $a_j = (j+1)/j^2$ (a partir de la a_1). *Comprovació*: Per a $n = 3$ ha d'escriure 2. 0.75 0.444444 (cada nombre en una línia).

Exercici 2.32
Feu un programa que demani un nombre natural n i escrigui els n primers múltiples positius de 7, tot començant pel 7.

Exercici 2.33
Feu un programa que demani a l'usuari un nombre natural i li digui si és primer o no (recordeu que un nombre natural és primer si no és divisible per cap nombre natural més petit que ell i més gran que 1). *Suggeriments*: És molt recomanable fer un esquema del programa en pseudocodi abans de començar a escriure'l en Fortran. Penseu que el programa ha de ser una transcripció del procés que seguiríem si ho féssim a mà: veure si el nombre és divisible per 2; si ho és ja podem afirmar que no és primer i, si no, repetirem el procés amb el 3, etc. Una vegada us funcioni el programa podeu optimitzar l'algorisme tenint en compte que un nombre n no pot ser divisible per un altre major que $n/2$.

Do aniuats

Un bloc d'instruccions `do - end do` pot incloure altres blocs `do - end do` sempre que s'utilitzin índexs diferents per a cada bloc —atès que l'índex d'un `do` no es pot modificar en el seu conjunt d'instruccions— i que cada bloc `do - end do` estigui contingut completament dins el bloc immediatament exterior. Es diu que els blocs interiors estan *aniuats* en els exteriors. Per tal de poder identificar amb facilitat els blocs d'instruccions de cada `do` és molt recomanable sagnar-los, de manera que els més interns es comencin a escriure més a la dreta.

Per exemple, el programa següent genera una llista de les variacions amb repetició dels dígits 1 a 9 agafats de dos en dos (11, 12, ..., 19, 21, ..., 29, ..., 99):

```
do i = 1, 9
  do j = 1, 9
    write (6,*) i, j
  end do
end do
end
```

En iniciar-se l'execució d'aquest programa l'índex `i` pren el valor 1 i s'executa el conjunt d'instruccions del corresponent `do - end do` (el més extern). Aquest conjunt s'inicia amb un segon `do` que assigna un 1 a l'índex j i escriu els valors de i i de j: 11.[19] A continuació s'efectua la segona iteració del `do - end do` més intern, que assigna un 2 a j i escriu 12, i així successivament fins a escriure el 19. Llavors s'acaba el `do - end do` més intern i s'inicia la segona iteració de l'extern: s'assigna un 2 a i i s'executa de nou tot el `do - end do` intern, que escriurà: 21, 22, ..., 29. D'aquesta manera, a cada iteració del `do - end do` extern s'executa completament l'intern, fins a arribar a l'última, que escriurà: 91, 92, ..., 99. La taula següent mostra, d'esquerra a dreta, els valors que van prenent les

[19] Com que per a cada variació s'han d'escriure dues variables, aquestes poden quedar separades per un o més espais en blanc, depenent de com interpreti cada compilador el format lliure. Això es pot evitar utilitzant el format d'escriptura 2I1 en comptes del lliure (vegeu la secció 3.4) o escrivint un enter de dos dígits construït a partir dels valors de i i de j: `write (6,*) 10*i+j`

variables d'aquest programa al llarg de l'execució:

i	1				2			...	9		
j	1	2	...	9	1	...	9	...	1	...	9
S'escriu	11	12	...	19	21	...	29	...	91	...	99

Observem que el nombre de vegades que s'executa la instrucció `write` és $9 \times 9 = 81$. Com que cada variació s'escriurà en una nova línia, la llista no cabrà en la consola i no la veurem sencera. En la secció 2.12 veurem com podem guardar-la en un fitxer per poder-la visualitzar després amb un editor de textos.[20]

Exercici 2.34
Feu un programa que generi una llista de les variacions amb repetició dels dígits 1 a 9 agafats de tres en tres (111, 112, ..., 119, 121, 122, ..., 129, ..., 999).

Exercici 2.35
Feu un programa que generi una llista de les variacions sense repetició dels dígits 1 a 9 agafats de dos en dos (12, 13, ..., 19, 21, 23, ..., 29, ..., 91, 92, ..., 98).

Exercici 2.36
Feu un programa que generi una llista de les combinacions (sense repetició) dels dígits 1 a 9 agafats de dos en dos (12, 13, ..., 19, 23, 24, ..., 29, ..., 78, 79, 89). *Suggeriment*: Fixeu-vos que el primer dígit pren els valors 1, 2, ..., 8, i el segon és sempre major que el primer.

Exercici 2.37
Feu un programa que trobi tots els nombres naturals inferiors a 1000 tals que la suma dels seus dígits sigui 22. *Resultat*: 499, 589, 598, 679, 688, 697, 769, 778, 787, 796, 859, 868, 877, 886, 895, 949, 958, 967, 976, 985 i 994.

Exercici 2.38
Feu un programa que trobi tots els nombres naturals de fins a tres dígits ijk (i, j, k enters $\in [0, 9]$) que siguin iguals a la suma dels cubs dels seus dígits ($i^3 + j^3 + k^3$). *Resultat*: 0, 1, 153, 370, 371 i 407.

2.10.2 Acumuladors

Una aplicació important del `do` amb índex és el càlcul de sumatoris i productoris. En ambdós casos es fa servir una variable a la qual s'assigna un valor inicial abans que comenci el procés iteratiu (s'*inicialitza*) i que va acumulant un terme del sumatori o del productori a cada iteració. En direm un *acumulador*.

Per exemple, per calcular la suma dels n primers nombres naturals ($S = \sum_{i=1}^{n} i$) podem utilitzar el programa següent:

```
write (6,*) "Quants nombres naturals vols que sumi?"
read (5,*) n
ns = 0
do i=1,n
  ns = ns + i
end do
```

[20] També es pot fer que s'aturi la presentació de resultats en la pantalla cada vegada que s'omple, tot executant el programa amb la instrucció: `./nomprograma | more`

```
write (6,*) "La suma es", ns
end
```

Per facilitar el seguiment d'aquest tipus d'algorismes —i de molts altres— és útil construir una taula que reculli, de dalt a baix, l'evolució al llarg del programa de les variables més importants; en el nostre cas, `ns` i `i`:

`ns`	`i`
0	1
$0 + 1 = 1$	2
$0 + 1 + 2 = 3$	3
$0 + 1 + 2 + 3 = 6$	...
...	$n - 1$
$0 + 1 + 2 + \ldots + (n-1) = \sum_{i=1}^{n-1} i$	n
$0 + 1 + 2 + \ldots + (n-1) + n = \sum_{i=1}^{n} i$	

Vegem com actua l'acumulador `ns`:

- abans que comencin les iteracions l'inicialitzem a zero;
- en la primera iteració `i` val 1, de manera que se suma 1 al valor inicial de `ns` (0) i es guarda la suma (1) en la mateixa variable `ns`;
- en la segona iteració se suma `i`, que ara val 2, al valor de `ns` (1) i es guarda la suma (3) novament en variable `ns`;
- ...
- en la darrera (n-èsima) iteració se suma n al valor de `ns` ($1 + 2 + \ldots +$ (n–1)) i es guarda la suma ($1 + 2 + \ldots + n$) en la variable `ns`.

En sortir del `do` el programa escriu l'últim valor que s'ha assignat a `ns`, és a dir, el valor de la suma $\sum_{i=1}^{n} i$.

És important tenir en compte que primer s'efectua l'operació que és a la dreta del signe '=' i després se n'assigna el resultat a la variable que és a l'esquerra, de manera que aquella operació s'efectuarà amb el valor que tenia `ns` en acabar la iteració anterior, i el seu resultat quedarà com a nou valor de `ns` al terme de la iteració actual. Observem que, si no haguéssim donat el valor inicial 0 a l'acumulador abans que comencin les iteracions, la primera d'aquestes agafaria el valor contingut en la zona de la memòria RAM que correspon a la variable *ns* i li sumaria un 1. Com que no sabríem què hi havia en aquesta zona de la RAM, el nou valor de *ns* seria imprevisible (encara que alguns compiladors assignen automàticament un zero a qualsevol variable que no hagi estat assignada pel programa).

Exercici 2.39
Feu un programa que demani a l'usuari un nombre natural n i calculi i escrigui el sumatori $\sum_{j=1}^{n} a_j$, on $a_j = (j+1)/j^2$. *Comprovació*: Per a $n = 3$ ha de donar 3,19444.

Exercici 2.40
Feu un programa que demani a l'usuari un nombre natural n i calculi i escrigui el factorial d'aquest nombre:

$$n! = 1 \times 2 \times 3 \times \ldots \times n = \prod_{j=1}^{n} j$$

(tingueu en compte que 0! = 1 per conveni). *Comprovació*: 8! = 40320. *Advertiment*: com que $n!$ creix molt de pressa amb n, supera de seguida el valor màxim que pot emmagatzemar-se en una variable entera (vegeu l'apartat 2.3.5), i en aquest cas donarà un resultat erroni. Comproveu-ho fent dues versions del programa; en la primera utilitzeu una variable entera per acumular els productes, i en la segona feu servir una variable real de precisió doble. Esbrineu fins a quin nombre funciona correctament la primera versió.

Exercici 2.41
Feu un programa que demani a l'usuari un nombre natural n i calculi i escrigui la suma dels nombres senars més petits o iguals que n; per exemple, si l'usuari introdueix un 5, el programa haurà d'escriure un 9 $(= 1 + 3 + 5)$.

Exercici 2.42
Feu un programa que demani a l'usuari un nombre natural n i calculi i escrigui la suma dels n primers nombres senars; per exemple, si l'usuari introdueix un 3, el programa haurà d'escriure un 9 $(= 1 + 3 + 5)$.

Exercici 2.43
Un professor té una pila d'exàmens corregits, amb notes de 0 a 10, i vol saber la nota mitjana de tots. Feu un programa que li demani el nombre d'exàmens i la nota de cadascun, i en calculi i n'escrigui la nota mitjana.

Exercici 2.44
Feu un programa que calculi el nombre de combinacions (sense repetició) que es poden formar amb m elements $(m \geq 1)$ en grups de n $(m \geq n \geq 1)$:

$$C_{m,n} = \binom{m}{n} = \frac{m!}{n!(m-n)!} = \frac{m \times (m-1) \times \ldots \times (m-n+1)}{n!}$$

Comprovació: Per a $m = 7$ i $n = 3$ ha de donar 35.

Exercici 2.45
Feu un programa que calculi el resultat del doble sumatori

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \frac{i+j}{i \times j} = \sum_{i=1}^{m} \left(\frac{i+1}{i \times 1} + \frac{i+2}{i \times 2} + \ldots + \frac{i+n}{i \times n} \right)$$

essent m i n dos nombres naturals que haurà d'introduir l'usuari. El programa ha d'escriure també el valor de $\sum_{j=1}^{n} \frac{i+j}{i \times j}$ per a cada valor de i. *Comprovació*: Per a $m = 3$ i $n = 4$ la suma total ha de donar 13,5833.

Exercici 2.46
Feu un programa que demani a l'usuari dos nombres naturals, m i n, avaluï l'operació següent:

$$\sum_{i=1}^{m} \left(\prod_{j=1}^{n} \frac{i}{j} \right) = \left(\frac{1}{1} \times \frac{1}{2} \times \ldots \times \frac{1}{n} \right) + \left(\frac{2}{1} \times \frac{2}{2} \times \ldots \times \frac{2}{n} \right) + \ldots + \left(\frac{m}{1} \times \frac{m}{2} \times \ldots \times \frac{m}{n} \right)$$

i n'escrigui el resultat. El programa ha d'escriure també el valor de $\prod_{j=1}^{n} \frac{i}{j}$ per a cada valor de i. *Comprovació* : Per a $m = 3$ i $n = 4$ la suma total ha de donar 4,08333.

2.10.3 *Do while*

En molts algorismes de tipus repetitiu no es coneix de bon principi el nombre d'iteracions que s'han de fer, però se sap quina condició s'ha de complir perquè el procés hagi convergit suficientment a la solució buscada. Per transcriure al llenguatge Fortran aquests algorismes s'utilitza el bloc d'instruccions `do while - end do`:[21]

```
do while (condició)
  conjunt d'instruccions
end do
```

En executar-se la instrucció `do while` s'avalua la condició dels parèntesis; si aquesta és certa, s'executa el conjunt d'instruccions que constitueixen una iteració, i si és falsa, es passa el control de l'execució a la instrucció següent a l'`end do`. Com en el cas del `do` amb índex, la instrucció `end do` retorna el control de l'execució a la instrucció `do while`.

Vegem-ne un primer exemple molt senzill: es tracta de modificar el fragment de programa de la pàgina 38 perquè, si l'usuari entra una pressió nul·la o negativa, li indiqui que aquesta dada ha de ser positiva i li doni l'oportunitat de corregir l'error; és a dir, li demani un nou valor per a la pressió:

```
...
write (6,*) "Entreu la pressio en bars"
read (5,*) p
do while (p .le. 0.0)
  write (6,*) "La pressio ha de ser > 0; entreu un nou valor"
  read (5,*) p
end do
...
```

Si l'usuari entra un valor positiu, la condició del `do while` serà falsa i no s'executarà cap vegada el conjunt d'instruccions del bloc, però si entra un valor nul o negatiu, es presentarà un missatge explicatiu i es tornarà a demanar la pressió. Si l'usuari és tan tossut que torna a entrar una dada incorrecta, el programa li tornarà a demanar la pressió, i així successivament fins que la dada entrada sigui positiva.

Exercici 2.47
Feu un programa que sumi un nombre no prefixat de nombres reals. El programa anirà demanant a l'usuari que entri nous valors fins que aquest entri un zero; llavors el programa deixarà de demanar dades i escriurà la suma dels valors entrats.

Exercici 2.48
Feu un programa que vagi demanant nombres reals a l'usuari fins que aquest entri un zero, sumi els nombres positius per una banda i els negatius per l'altra, i escrigui les dues sumes.

Exercici 2.49
Feu un programa que demani un nombre natural n i, emprant la instrucció `do while`, escrigui els múltiples positius de 7 més petits que n, tot començant pel 7.

Considerem una aplicació molt interessant del `do while`: la resolució iterativa d'equacions. Ho il·lustrarem mitjançant un algorisme iteratiu per calcular arrels quadrades fent només operacions aritmètiques (suma i divisió). Aquest és un exemple més acadèmic que útil, atès que el Fortran disposa de la funció incorporada `sqrt` per obtenir arrels quadrades (vegeu la secció 2.6); l'algorisme, però, podria ser útil si es volgués fer el càlcul amb una calculadora que no tingués la funció arrel. L'arrel quadrada x d'un nombre positiu a compleix l'equació:

$$x = \frac{x + \frac{a}{x}}{2} \tag{2.1}$$

[21] Aquesta construcció no és estàndard del Fortran 77, però, atesa la seva utilitat, gairebé tots els compiladors l'han incorporat.

ja que, si multipliquem els dos membres per $2x$ i simplifiquem el resultat, s'obté:

$$x^2 = a$$

Per obtenir el valor de x que compleix l'equació (2.1) podem substituir una primera aproximació (x_0) de l'arrel buscada en el membre dret d'aquella equació i calcular el valor x_1 que pren aquest membre:

$$x_1 = \frac{x_0 + \frac{a}{x_0}}{2}$$

Si x_1 coincidís amb x_0 aquest valor seria la solució x buscada. El més probable, però, és que x_1 sigui diferent de x_0, la qual cosa indicarà que aquest valor no compleix l'equació (2.1). Llavors agafarem x_1 com a nova aproximació a x i el substituirem al lloc de x_0 en el membre dret de l'equació anterior per calcular un nou valor de x_1. Aquest cicle es repetirà de manera iterativa fins que el valor substituït al membre dret (x_0) coincideixi amb el resultat d'avaluar aquest membre (x_1), la qual cosa voldrà dir que aquest valor compleix l'equació (2.1); és a dir, que és igual a $\sqrt{a}$. Com a valor de prova inicial per començar les iteracions (la *llavor* del procés iteratiu) utilitzarem el mateix nombre a.[22] El programa que implementa aquest algorisme podria ser:

```
write (6,*) "De quin nombre positiu voleu calcular l'arrel?"
read (5,*) a
x0 = a
x1 = (x0+a/x0)/2
do while (x1 .ne. x0)
  x0 = x1
  x1 = (x0+a/x0)/2
end do
write (6,*) "L'arrel de", a, " es", x1
end
```

Si l'usuari entrés un 1 com a dada, la llavor seria directament l'arrel buscada ($x_0 = a = \sqrt{a} = x_1$). Aleshores la condició del `do while` seria falsa (`x1` prendria el mateix valor que `x0`) i no s'executaria el bloc d'instruccions corresponent; el programa passaria directament a escriure el resultat. Per a valors de `a` diferents d'1 la instrucció de la quarta línia assignarà a `x1` un resultat diferent de `x0`, de manera que la condició serà certa i s'executarà el bloc d'instruccions. Aquest inclou la transferència del valor de `x1` a la variable `x0` seguida del càlcul d'un nou valor per a `x1` a partir del nou valor de `x0`. L'`end do` retorna el control del programa a la instrucció `do while` i, si `x1` és encara diferent de `x0`, es repeteix el procés. Observeu que la condició que apareix en la instrucció `do while` ha de ser sempre la contraria a la que determina la convergència del procés iteratiu, ja que les iteracions s'han de succeir mentre el criteri de convergència no s'hagi assolit.

Els algorismes d'aquest tipus poden presentar problemes de convergència: atès que les operacions amb nombre reals comporten un petit error, no és convenient utilitzar com a criteri de convergència la igualtat entre x_0 i x_1 (pot ser que mai no arribin a ser exactament iguals); una alternativa millor és exigir que ambdues variables difereixin menys d'una xifra ε una mica més gran que l'error implícit en la codificació de les dades; per exemple, 10^{-6} (teclejarem `1e-6`) si treballem amb reals de precisió senzilla. Com que no sabem si x_1 serà major o menor que x_0 s'haurà d'utilitzar el valor absolut de la seva diferència per establir la condició de convergència: $|x_1 - x_0| \leq \varepsilon$.

Exercici 2.50
Modifiqueu el programa de l'exemple anterior per tal que llegeixi el valor d'una variable `eps` i aturi les iteracions quan $|x_1 - x_0|$ sigui més petit o igual que aquest valor.

[22] $\sqrt{a}$ és menor o major que a segons que aquest nombre sigui major o menor que 1, de manera que a és una aproximació de compromís a $\sqrt{a}$ acceptable per a qualsevol a. Si desenvolupem $\sqrt{a}$ entorn de $a = 1$ obtenim llavors més acurades, com ara $(1 + a)/2$. Una altra opció és demanar a l'usuari que entri un valor aproximat de $\sqrt{a}$.

Exercici 2.51
Feu un programa que calculi i escrigui el volum V en litres que ocuparan n mols d'un gas que compleix l'equació d'estat de van der Waals:

$$\left(P + \frac{an^2}{V^2}\right)(V - nb) = nRT$$

a una temperatura en graus Celsius t i una pressió en bars P ($R = 0{,}08314472$ bar l K^{-1} mol^{-1} i $T/\mathrm{K} = t/^\circ\mathrm{C} + 273{,}15$). Com que en aquesta equació no es pot aïllar la variable V dissenyeu un algorisme iteratiu a partir de l'equació expressada de la manera següent:

$$V = \frac{nRT}{P + \frac{an^2}{V^2}} + nb$$

Com a llavor per al procés iteratiu podeu emprar el volum que correspondria a un gas ideal: $V_{id} = nRT/P$. Feu que el programa escrigui també aquest volum per tal que l'usuari pugui apreciar el grau de no-idealitat del gas. *Comprovació*: Per a 1,5 mols de N_2 ($a = 1{,}370$ l^2 bar mol^{-2}, $b = 0{,}0387$ l mol^{-1}) a una temperatura de 20°C i 5,5 bars de pressió s'obté un volum ideal de 6,64742 litres i un volum de van der Waals de 6,62157 litres.

Exercici 2.52
Feu un programa que trobi una solució en radians de l'equació $x = 1/\sqrt{1 + \tan^2(x)}]$ amb una precisió ε a partir d'una llavor x_0, la qual ha d'estar compresa entre 0 i 1 (aquesta equació apareix en el càlcul de les energies dels estats quàntics lligats d'un pou de potencial finit). *Resultat*: $x = 0{,}739085$ (només n'hi ha una).

2.10.4 Comptadors

A l'hora d'executar-se un programa que conté un `do while` pot plantejar-se un problema: si la condició d'aquesta instrucció no deixa de complir-se mai, les iteracions no s'aturaran. Per exemple, pot ocórrer que la resolució iterativa d'una equació no convergeixi per a valors de la llavor molt allunyats de la solució. Si això passa (l'ordinador semblarà que no fa res, però estarà iterant com un boig), podem forçar la interrupció de l'execució (*avortar* el programa) prement la tecla ctrl i, sense deixar-la anar, la tecla C. Una solució més elegant consisteix a introduir un *comptador d'iteracions* en el conjunt d'instruccions del bloc `do while` - `end do` i una estructura alternativa que aturi l'execució si el nombre d'iteracions arriba a un límit preestablert, tot explicant a l'usuari el que ha passat.

El comptador és una variable entera a la qual s'ha d'assignar el valor zero abans que comenci el procés iteratiu (igual que s'ha de posar a zero el comptaquilòmetres parcial d'un cotxe quan es vol mesurar una distància per carretera) i s'incrementa en 1 a cada iteració (el mateix que fa el comptaquilòmetres cada vegada que s'ha recorregut 1 km). És un cas particular d'acumulador que s'inicialitza a zero i va acumulant uns. Vegem com es pot introduir un comptador en el programa presentat en l'apartat anterior (vegeu també l'exercici 2.50):[23]

```
write (6,*) "De quin nombre positiu voleu calcular l'arrel?"
read (5,*) a
write (6,*) "Amb quina precisio voleu calcular l'arrel?"
read (5,*) eps
x0 = a
x1 = (x0+a/x0)/2
iter = 0
```

[23] De fet, aquest algorisme convergeix molt bé, i no caldria introduir cap comptador.

```
do while (abs(x1-x0) .gt. eps)
  if (iter .eq. 1000) then
    write (6,*) "Sembla que el proces no convergeix"
    write (6,*) "(s'han fet 1000 iteracions)"
    stop
  else
    x0 = x1
    x1 = (x0+a/x0)/2
    iter = iter+1
  end if
end do
write (6,*) "L'arrel de", a, " es", x1
end
```

Exercici 2.53
Modifiqueu el programa de l'exemple anterior perquè demani a l'usuari el nombre màxim d'iteracions a partir del qual s'atura el procés iteratiu. Proveu el funcionament correcte del programa entrant un valor molt petit per al nombre màxim d'iteracions.

Exercici 2.54
Modifiqueu el programa de l'exercici 2.51 perquè escrigui l'aproximació calculada per al volum del gas a cada iteració, s'aturi si el nombre d'iteracions passa d'un valor màxim prèviament demanat a l'usuari, i escrigui el nombre total d'iteracions efectuades.

Exercici 2.55
El nombre π es pot calcular a partir de la sèrie $\sum_{j=1}^{\infty}(1/j^2) = \pi^2/6$. Com que no podem fer un programa que sumi un nombre infinit de termes, negligirem els menors que un valor ε suficientment petit. El càlcul de la suma truncada es pot programar de dues maneres:
i) calculant primer el valor de j per a l'últim terme que volem sumar ($1/j^2 \approx \varepsilon$) i fent servir un `do` amb índex per avaluar la suma;
ii) fent la suma amb un `do while` que aturi l'acumulació de termes quan aquests siguin més petits que ε.
Feu un programa que calculi un valor aproximat del nombre π de la primera manera i, després, un altre que ho faci de la segona. Feu servir variables reals de precisió doble per poder obtenir valors acurats de π. El valor de ε s'ha de demanar a l'usuari com a dada.
Comprovació: $\pi = 3{,}14159265358979323846\ldots$

Exercici 2.56
El mètode numèric de Newton-Raphson per buscar solucions reals d'una equació del tipus $f(x) = 0$ es basa en l'algorisme iteratiu següent (vegeu la figura 2.1):
i) s'escull un valor inicial x_0 com a primera estimació de la solució;
ii) es calcula un valor nou x_1 a partir de l'anterior emprant l'equació: $x_1 = x_0 - f(x_0)/f'(x_0)$; si $|f(x_1)| < \varepsilon$ (un nombre una mica major que la imprecisió de les variables utilitzades), donem per acabat el procés amb x_1 com a solució i, si no, s'agafa el valor de x_1 com a nova estimació de la solució —és a dir, com a nou valor de x_0— i es repeteix el càlcul de $f(x_0)$, $f'(x_0)$, x_1 i $f(x_1)$.
a) Feu un programa que busqui una solució real de l'equació de tercer grau $ax^3 + bx^2 + cx + d = 0$ emprant el mètode de Newton-Raphson. Les dades a introduir per l'usuari són els coeficients a, b, c i d, una primera estimació de la solució buscada (x_0) i el valor de ε. És convenient utilitzar variables de precisió doble per minimitzar l'error numèric. *Comprovació* : Les solucions de l'equació $(x-1)(x-2)(x-3) = x^3 - 6x^2 + 11x - 6 = 0$ són 1, 2 i 3. Mireu de trobar-les utilitzant diferents llavors properes a cada solució.

b) Introduïu un comptador d'iteracions i feu que el procés iteratiu s'interrompi si el nombre d'iteracions passa d'un cert valor màxim.
c) L'algorisme utilitzat en l'apartat anterior presentarà problemes si la derivada de la funció en algun dels punts x_0 és molt petita, ja que aquesta derivada apareix al denominador de l'expressió que determina x_1. Modifiqueu el programa perquè emeti un missatge explicatiu i s'aturi si el valor absolut de la derivada és inferior a ε.
Comprovació: Per a les dades indicades en l'apartat a), amb $x_0 = 1,42264973$ i $\varepsilon = 10^{-8}$ el programa s'ha d'aturar la primera vegada que es comprovi el valor de la derivada.

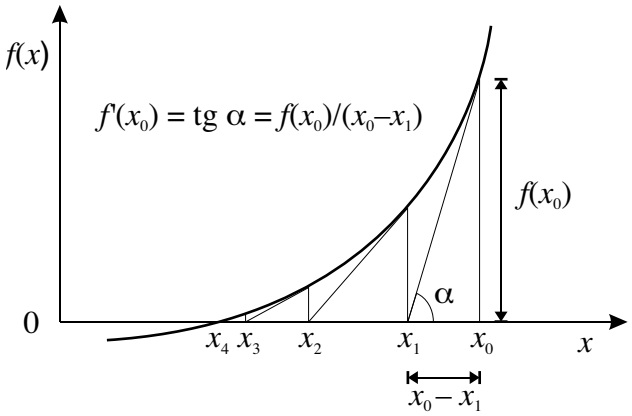

Figura 2.1: Mètode de Newton-Raphson.

Exercici 2.57
Un professor té una pila d'exàmens corregits, amb notes de 0 a 10, i vol saber quants estan aprovats i quants estan suspesos.
a) Feu un programa que li vagi demanant notes fins que entri un valor negatiu —la qual cosa indicarà que s'ha acabat la pila— i n'escrigui el nombre d'aprovats i el de suspesos.
b) Modifiqueu el programa anterior perquè en doni també la nota mitjana global.

2.11 Variables indexades

Tant en el llenguatge matemàtic com en un de programació pot interessar identificar amb un nom un conjunt de dades del mateix tipus; per exemple, les qualificacions obtingudes pels alumnes presentats a un examen, els components d'un vector d'un espai de dimensió n, una matriu real de dimensions m i n, etc. En el llenguatge matemàtic, cada element d'un d'aquests conjunts se sol identificar afegint un o més subíndexs al nom del conjunt; per exemple, el component V_2 d'un vector $\vec{V}$, l'element A_{ij} d'una matriu A, etc. En els llenguatges de programació, un conjunt de variables del mateix tipus agrupades amb un mateix nom s'anomena una *variable indexada*. Aquestes s'anomenen també *vectors* quan tenen un sol índex, i *matrius*, si en tenen dos. Com que la sintaxi del Fortran no permet utilitzar subíndexs, els índexs que identifiquen cada element s'escriuen entre parèntesi: el component `v(2)` del vector `v`, l'element `a(i,j)` de la matriu `a`, etc. Els noms de les variables indexades, com els de les no indexades o *escalars* (vegeu l'apartat 2.3.4), poden tenir d'1 a 6 caràcters alfanumèrics (lletres de l'alfabet anglès i/o dígits numèrics) dels quals el primer ha de ser forçosament una lletra.

2.11.1 Vectors

Imaginem que volem fer un programa que llegeixi les notes obtingudes per un grup d'alumnes en un examen i que, després de llegir-les totes, permeti fer algun tractament amb elles, com ara el càlcul de la nota mitjana i la desviació típica, treure una llista d'aprovats, etc. Utilitzar una variable diferent per a cada nota seria poc pràctic si són molts alumnes. És millor fer servir un vector amb

tants components com alumnes s'han presentat a l'examen, de manera que cada nota es guardi en un component del vector. Per exemple, si anomenem `nota` al vector, la nota del primer alumne s'emmagatzemarà en la variable `nota(1)`, que és el primer component (o element) del vector, la nota del segon alumne es guardarà en la variable `nota(2)`, etc. L'índex ha de prendre un valor enter, però pot ser una constant (`nota(2)`), una variable (`nota(i)`) o una expressió (`nota(2*i+1)`).

Qualsevol variable no declarada es considera no indexada, de manera que les variables indexades s'han de declarar sempre. Això es pot fet mitjançant una instrucció `dimension`:

```
dimension vect1(dim1), vect2(dim2) ...
```

on *dim1*, *dim2* ... són constants enteres que declaren *vect1* com un vector de dimensió *dim1*, *vect2* com un vector de dimensió *dim2*, etc.

La lectura d'un vector és un procés repetitiu, ja que s'han de llegir un a un els valors dels components. Vegem un exemple de programa que llegeix les notes d'un grup de 50 alumnes i, després de llegir-les totes, escriu una llista amb els aprovats:

```
dimension nota(50)
write (6,*) "Entreu la nota de cada alumne per ordre de llista"
write (6,*) "(premeu Retorn despres de cada nota)"
do i=1,50
  read (5,*) nota(i)
end do
write (6,*) "Llista d'aprovats:"
do i=1,50
  if (nota(i) .ge. 5) write (6,*) "L'alumne",i," ha tret un",
$  nota(i)
end do
end
```

Observeu que l'usuari ha de prémer la tecla de retorn després de teclejar cada nota, ja que cada vegada que s'executa la instrucció `read` es llegeix un sol component del vector (vegeu la secció 2.4).

No s'ha de confondre l'índex d'un component, que indica de quin alumne es tracta, amb el valor guardat en el component, que és la nota de l'alumne. La distinció entre aquests dos valors es fa ben palesa en la instrucció d'escriptura del programa anterior. Així, si en executar el programa l'usuari entra les notes 5, 4, 8, ..., 9 la instrucció `read` guardarà aquests valors en els diferents components del vector `nota`, tal con s'indica en l'esquema següent:

`i` ⟶	1	2	3	...	50
`nota(i)` ⟶	5	4	8	...	9

i el programa escriurà:

```
L'alumne 1 ha tret un 5
L'alumne 3 ha tret un 8
...
L'alumne 50 ha tret un 9
```

D'acord amb els convenis sobre noms de variables (apartat 2.3.5), els elements d'un vector seran nombres enters si el nom del vector comença per qualsevol de les lletres 'i' a 'n', i reals si comença per una altra lletra, sempre que no s'indiqui altrament mitjançant alguna instrucció de declaració de tipus de dades explícita (`integer`, `real`, etc.) o implícita (`implicit double precision`, etc.). En l'exemple anterior, si les notes tinguessin decimals hauríem d'afegir la instrucció

```
real nota
```

o bé donar al vector un nom que comenci per una lletra que correspongui a una variable real; per exemple, `xnota`.

També es pot declarar el tipus i el nombre de components del vector en una sola instrucció de declaració explícita de tipus de dades, de manera que les dues instruccions de declaració del programa (`dimension nota(50)` i `real nota`) es podrien substituir per una d'equivalent:

```
real nota(50)
```

El programa anterior s'hauria de modificar per poder aplicar-lo a un grup amb un nombre d'alumnes diferent de 50. Això es pot evitar declarant els vectors amb una dimensió prou gran per als grups que preveiem considerar. Llavors, en executar-se el programa s'haurà de demanar a l'usuari el nombre d'alumnes que té el grup considerat i restringir el tractament a aquest nombre de notes. Per incloure aquesta modificació en el programa suposarem que el nombre d'alumnes no pot passar de 150; d'altra banda, farem que es llegeixi també el cognom de cada alumne i s'escrigui en la llista d'aprovats:[24]

```
real nota(150)
character*12 cognom(150)
write (6,*) "Entreu el nombre d'alumnes del grup"
write (6,*) "(no pot ser superior a 150)"
read (5,*) numal
write (6,*) "Entreu el cognom de cada alumne i la seva nota"
write (6,*) "(premeu Retorn despres de cada nota)"
do i=1,numal
  read (5,*) cognom(i), nota(i)
end do
write (6,*) "Llista d'aprovats:"
do i=1,numal
  if (nota(i) .ge. 5) write (6,*) cognom(i)," ha tret un",
$  nota(i)
end do
end
```

Observem que si executem aquest programa per a un grup de 50 alumnes s'utilitzaran només 50 dels 150 elements de cada vector. Podríem pensar a utilitzar vectors de dimensió ajustada al nombre d'alumnes en cada execució, però el Fortran 77 no permet usar variables per dimensionar vectors.[25] Això és degut al fet que el compilador analitza les instruccions de declaració per saber quanta memòria RAM s'haurà de reservar en començar l'execució del programa, i si posem

```
real nota(numal)
read (5,*) numal
```

el compilador no pot saber el valor que donarà l'usuari a la variable `numal`. Tampoc no serveix fer la lectura de la variable `numal` abans de la declaració:

```
read (5,*) numal
real nota(numal)
```

ja que les instruccions de declaració han d'anar abans que les executables. Per les mateixes raons no es pot establir la dimensió mitjançant una instrucció d'assignació:

```
real nota(numal)
numal = 50
```

o

```
numal = 50
real nota(numal)
```

El que sí es pot fer és declarar prèviament una constant figurativa i fer-la servir després per dimensionar un o més vectors. Per exemple, si anomenem `maxal` a aquesta constant, podem posar:

[24] Si el cognom introduït és compost i conté espais en blanc s'ha de posar entre cometes.

[25] Les versions més recents del Fortran i altres llenguatges de programació, permeten utilitzar variables indexades de dimensió variable durant l'execució del programa. Això s'anomena *assignació dinàmica de la memòria*, i és una pràctica poc recomanable per a programadors principiants.

```
parameter (maxal=150)
real nota(maxal)
character*12 cognom(maxal)
```

D'aquesta manera, si es vol modificar la dimensió d'aquests vectors només s'haurà de canviar la instrucció `parameter` i recompilar el programa.

Exercici 2.58
Indiqueu la memòria RAM, expressada en bytes i en kiB, que ocuparan durant l'execució les variables declarades explícitament en el fragment de programa següent:

```
implicit double precision (a-h)
dimension j5(5), tab(1000), cx(50)
integer a(15), b1, b2, b3
real mol(25), g
character*11 nom(20), cognom(20)
```

Solució: 5036 bytes $\approx$ 4,918 kiB.

Exercici 2.59
Modifiqueu el programa de la pàgina 55 perquè calculi i escrigui també el nombre d'alumnes aprovats i el de suspesos.

Exercici 2.60
Feu un programa que llegeixi els components de dos vectors reals $\vec{V}$ i $\vec{W}$ de la mateixa dimensió n, la qual s'haurà d'entrar com a dada amb un màxim de 100, i calculi i escrigui els components del vector suma d'aquells dos: $\vec{S} = \vec{V} + \vec{W}$ (tingueu en compte que $S_1 = V_1 + W_1,\ \ldots, S_n = V_n + W_n$; per exemple, $(1,2,3) + (-3,-2,-1) = (-2,0,2)$).

Exercici 2.61
Escriviu un programa que llegeixi el nombre de components (n) d'un vector real $\vec{V}$ en una base ortonormal (amb un màxim de 100) i els valors d'aquests, i calculi i escrigui el mòdul del vector: $\left\|\vec{V}\right\| = \sqrt{V_1^2 + V_2^2 + \ \ldots + V_n^2} = \sqrt{\sum_{i=1}^{n} V_i^2}$. *Comprovació*: El mòdul de $(3,2,5)$ és 6,16441.

Exercici 2.62
Escriviu un programa que llegeixi els components de dos vectors reals $\vec{V}$ i $\vec{W}$ de la mateixa dimensió n (la qual s'haurà d'entrar com a dada, amb un màxim de 100) en una base ortonormal i calculi i escrigui el producte escalar d'aquells: $\vec{V}\cdot\vec{W} = V_1W_1 + V_2W_2 + \ldots + V_nW_n = \sum_{i=1}^{n} V_iW_i$. *Comprovació*: El producte escalar de $(1,2,3)$ i $(-3,-2,-1)$ és -10.

Exercici 2.63
Feu un programa que llegeixi el nombre d'alumnes presentats a un examen ($numal$) i les seves notes $nota(1),\ \ldots, nota(numal)$, i calculi i escrigui la nota mitjana $\overline{nota} = \frac{\sum_{i=1}^{numal} nota(i)}{numal}$ i la desviació típica de les notes: $\Delta nota = \sqrt{\frac{\sum_{i=1}^{numal}\left(nota(i)-\overline{nota}\right)^2}{numal}}$. *Comprovació*: La mitjana de les cinc notes 1,80, 7,20, 5,40, 9,80 i 8,00 és 6,44 i la seva desviació típica és, aproximadament, 2,71.

Exercici 2.64
Feu un programa que llegeixi el nombre d'alumnes presentats a un examen, els seus cognoms i les seves notes, i faci una llista amb els cognoms i notes dels aprovats (nota $\in [5, 7)$), una altra amb els dels notables (nota $\in [7, 9)$), i una tercera amb els dels excel·lents (nota $\in [9, 10]$). El programa ha d'escriure també el nombre d'aprovats, el de notables i el d'excel·lents.

Exercici 2.65
La distància entre dos vectors reals $\vec{a}$ i $\vec{b}$ de n components es defineix com $d = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$. Feu un programa que llegeixi el nombre de components de dos vectors i els valors d'aquests i calculi la distància entre ells. *Comprovació*: Distància entre $(0, 1, 2)$ i $(-1, -2, -3) = 5{,}91608$.

2.11.2 *Do* implícit

Les instruccions del programa mostrat a la pàgina 54 que llegeixen els components del vector `nota` exigeixen que es premi la tecla de retorn després de teclejar cada valor. Això és degut al fet que, cada vegada que s'executa la instrucció `read`, l'ordinador no passa a la instrucció següent fins que s'hagi entrat la dada demanada. Si volguéssim entrar cada nota a continuació de l'anterior (separada per una coma o un espai en blanc), sense haver de prémer la tecla Retorn fins al final, hauríem de substituir l'estructura repetitiva:

```
do i=1,numal
  read (5,*) nota(i)
end do
```

per la instrucció:

```
read (5,*) (nota(i), i=1,numal)
```

La construcció entre parèntesis s'anomena *do implícit*. Encara que no hi apareix la paraula *do*, aquesta construcció funciona de manera molt semblant al `do` (explícit) amb índex: la lectura de la variable `nota(i)` es repetirà tot donant a l'índex `i` valors enters consecutius des de 1 fins a `numal`. La principal diferència entre ambdues construccions és que en el `do` explícit es fan `numal` execucions de la instrucció `read` i el do implícit és una opció d'una instrucció `read` que s'executa una única vegada i requerirà prémer un sol cop la tecla de retorn. Observem que això és semblant al que succeeix quan es llegeixen diverses variables `a`, `b` ... no indexades seguides: si es fa amb una única instrucció `read`:

```
read (5,*) a, b ...
```

es poden entrar els seus valors l'un a continuació de l'altre i prémer Retorn al final; en canvi, si s'utilitza una instrucció `read` per a cada variable:

```
read (5,*) a
read (5,*) b
 ...
```

s'haurà de prémer Retorn després de cada valor.

El *do* implícit afecta totes les variables incloses en el parèntesi abans de l'*índex* i, com l'explícit, admet l'especificació d'un increment, que haurà d'anar separat per una coma del valor final. Així, la instrucció:

```
read (5,*) (vect1(índex), vect2(índex), ..., índex=valin,valfin,incr)
```

llegirà les dades *vect1*(1), *vect2*(1), ..., *vect1*(1+*incr*), *vect2*(1+*incr*) ... seguides. Per exemple, si substituïm les instruccions de lectura de dades del programa de la pàgina 55 per

```
read (5,*) (cognom(i), nota(i), i=1,numal)
```

podrem introduir les parelles de dades cognom-nota de tots els alumnes seguides (tot i que, en aquest cas, és millor no emprar el *do* implícit i llegir-les en línies diferents).

La instrucció `write` també admet el *do* implícit com a opció; així, la instrucció

```
write (6,*) (vect1(índex), vect2(índex), ..., índex=valin,valfin,incr)
```

escriurà els valors de les variables *vect1*(1), *vect2* (1), ...,*vect1*(1+*incr*), *vect2*(1+*incr*) ... en la mateixa línia si hi caben —si no, es continuarà a la línia següent—, a diferència de les instruccions:

```
do índex=valin,valfin,incr
  write (6,*) vect1(índex), vect2(índex) ...
end do
```

que escriurien en una línia diferent els valors corresponents a cada valor de l'índex.

Exercici 2.66
Modifiqueu el programa de l'exercici 2.60 perquè llegeixi tots els components de cada vector en una mateixa línia i escrigui en una sola línia els del vector suma.

S'ha de tenir present que el *do* implícit és una opció de les instruccions `read` i `write`, i no es pot utilitzar fora d'aquestes. Per exemple, no es pot multiplicar per 2 els n elements d'un vector amb una instrucció del tipus:

```
(vec(i) = 2*vec(i), i=1,n)
```

la qual produiria un error de sintaxi. Una forma correcta de fer-ho seria:

```
do i=1,n
  vec(i) = 2*vec(i)
end do
```

Exercici 2.67
Feu un programa que demani un nombre natural expressat en notació decimal i l'escrigui en notació binària (vegeu la secció 1.3). *Suggeriment*: Per presentar el resultat podeu utilitzar un vector de 31 components enters, cada un dels quals contindrà un dígit del nombre binari, i escriure'l fent servir un *do* implícit amb índex decreixent des de 31 fins a 1. *Comprovació*: $13)_{10} = 1101)_2$; $0)_{10} = 0)_2$.

2.11.3 Matrius

Com els vectors, les variables indexades amb dos índexs (matrius) s'han de declarar sempre amb una instrucció `dimension` o amb una instrucció de declaració explícita de tipus de dades (`integer`, `real`, etc.) en la qual s'indiqui el nom de cada matriu i, entre parèntesis, els nombres de files i de columnes separats per una coma:

```
dimension mat1(nfil1,ncol1), mat2(nfil2,ncol2) ...
real mat1a(nfil1a,ncol1a), mat2a(nfil2a,ncol2a) ...
```

Per exemple, podem declarar una matriu A de 4 files i 3 columnes:

$$\mathsf{A} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{pmatrix}$$

mijançant la instrucció

```
real a(4,3)
```

Un element d'aquesta matriu, com ara el A_{23}, es pot identificar en el programa amb la notació `a(2,3)`.

Per fer qualsevol tractament amb tots els elements d'una matriu haurem d'utilitzar dues instruccions *do*, una per recórrer l'índex de les files i una altra per al de les columnes. Així, per llegir els elements d'una matriu de 4 files i 3 columnes podem utilitzar les instruccions següents:

```
dimension mat(4,3)
write (6,*) "Entreu els elements de la matriu per files"
write (6,*) "(premeu Retorn despres de cada element)"
do i=1,4
  do j=1,3
    read (5,*) mat(i,j)
  end do
end do
```

Observeu que, en aquest fragment de programa, cada vegada que s'executa la instrucció `read` es llegeix un sol element; per això s'ha de prémer la tecla de retorn després d'entrar cada dada. Per exemple, si l'usuari vol entrar la matriu

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

haurà d'escriure un '1' i prémer Retorn, escriure un '2' i prémer Retorn, escriure un '3' i prémer Retorn, escriure un '4' i prémer Retorn; així fins el 12. Si se substituís el `do` intern per un d'implícit es podrien entrar tots el elements d'una mateixa fila de cop —separats per comes o espais en blanc— i prémer després la tecla de retorn.[26]. Per exemple:

```
write (6,*) "Entreu els elements de la matriu per files"
write (6,*) "(premeu Retorn al final de cada fila)"
do i=1,4
  read (5,*) (mat(i,j), j=1,3)
end do
```

de manera que l'usuari escriuria 1,2,3 o 1 2 3 i premeria Retorn, etc.

Ha de quedar clar que els noms de les variables enteres que s'utilitzen com a índexs són irrellevants; el que és important és tenir present que el primer índex indica la fila i el segon, la columna. Així, la lectura de la matriu en l'exemple anterior funcionaria exactament de la mateixa manera si empréssim la variable j per identificar les files i la i per a les columnes:

```
do j=1,4
  read (5,*) (mat(j,i), i=1,3)
end do
```

La diferència entre els *do* implícit i explícit es fa especialment palesa quan s'escriu una matriu: la utilització de dos *do* explícits fa que cada vegada que s'executa la instrucció `write` s'escrigui un sol valor i, en conseqüència, cada un d'aquests s'escriurà en una línia diferent. Així, si s'escriu la matriu de l'exemple anterior fent servir les instruccions:

```
do i=1,4
  do j=1,3
    write (6,*) mat(i,j)
  end do
end do
```

el que se presentarà en el monitor és:

[26] Fins i tot, es podrien llegir tots els elements de la matriu de cop utilitzant dos *do* implícits aniuats:

```
read (5,*) ((mat(i,j), j=1,3), i=1,4)
```

tot i que la introducció de les dades resultaria força menys intuïtiva.

```
1
2
3
4
...
12
```

En canvi, si substituïm el `do` intern per un d'implícit:

```
do i=1,4
  write (6,*) (mat(i,j), j=1,3)
end do
```

s'escriuran els elements de cada fila en una mateixa línia (direm que la matriu s'ha escrit *per files*):

```
1 2 3
4 5 6
7 8 9
10 11 12
```

la qual cosa permetrà apreciar millor l'estructura matricial de les dades.

El mateix que s'ha dit en la pàgina 55 en relació a com fer que un programa serveixi per a vectors de diferents mides es pot aplicar a les matrius (vegeu l'exercici 2.69).

Exercici 2.68
Feu un programa que demani els elements d'una matriu quadrada de 4 files i 3 columnes, multipliqui per 2 la matriu (és a dir, cadascun dels seus elements) i escrigui la matriu resultant per files.

Exercici 2.69
a) Feu un programa que llegeixi dues matrius A i B de 4 files i 3 columnes, les sumi ($C_{ij} = A_{ij} + B_{ij}$) i escrigui la matriu suma (C) per files.
b) Modifiqueu el programa anterior perquè accepti matrius de dimensions qualssevol (fins a un màxim de 100×100), de manera que l'usuari hagi d'entrar el nombres de files i el de columnes de les matrius que vol sumar.

Exercici 2.70
Feu un programa que demani el nombre de files n d'una matriu quadrada A, assigni un 1 als elements diagonals (A_{ij} amb $i = j$) i un 0 als no diagonals (A_{ij} amb $i \neq j$), i escrigui la matriu resultant. És a dir, es tracta de construir i escriure la matriu identitat de dimensions $n \times n$.

Exercici 2.71
Feu un programa que llegeixi el nombre de files (m) i el de columnes (n) d'una matriu A i els seus elements, calculi una nova matriu B que sigui la transposada de A ($B = A^t$) i l'escrigui per files. Recordeu que els elements de la matriu transposada d'una matriu A estan relacionats amb els d'aquesta de la manera següent: $B_{ji} = A_{ij}$; dit d'una altra manera, les files de B són les columnes de A i viceversa. Utilitzeu una matriu no quadrada per comprovar que el programa funciona correctament quan els nombres de files i de columnes són diferents.

Exercici 2.72
Feu un programa que llegeixi una matriu quadrada A de dimensions $n \times n$, essent n una dada a introduir per l'usuari, multipliqui per 2 els elements de la seva diagonal principal (A_{11}, A_{22}, ..., A_{nn}) i escrigui la matriu resultant per files.

Exercici 2.73
Feu un programa que llegeixi una matriu quadrada A de dimensions $n \times n$, essent n una dada a introduir per l'usuari, calculi la seva traça i l'escrigui. Recordeu que la traça d'una matriu és la suma dels elements de la diagonal principal: $\mathrm{tr}A = A_{11} + A_{22} + \ldots + A_{nn} = \sum_{i=1}^{n} A_{ii}$.

Exercici 2.74
Feu un programa que llegeixi les dimensions m i n d'una matriu MAT de nombres enters i els seus elements, escrigui la matriu per files, i ens indiqui quants dels seus elements són positius, quants són zero i quants són negatius.

Considerem el problema següent: volem fer un programa que llegeixi els cognoms dels alumnes presentats a un examen que consta de 10 qüestions i les notes que han tret a cada qüestió. Els noms es guardaran en un vector que anomenarem `cognom` i les qualificacions, en una matriu anomenada `notes` que tingui tantes files com alumnes (`numal`) i tantes columnes com qüestions; la primera fila contindrà les notes del primer alumne, la segona fila contindrà les del segon, etc.:

$$\begin{array}{rcl} \text{notes de l'al. } 1 & \rightarrow & \left(\begin{array}{cccc} \texttt{notes(1,1)} & \texttt{notes(1,2)} & \ldots & \texttt{notes(1,10)} \\ \texttt{notes(2,1)} & \texttt{notes(2,2)} & \ldots & \texttt{notes(2,10)} \\ \ldots & \ldots & \ldots & \ldots \\ \texttt{notes(numal,1)} & \texttt{notes(numal,2)} & \ldots & \texttt{notes(numal,10)} \end{array} \right. \\ \text{notes de l'al. } 2 & \rightarrow & \\ \ldots & & \\ \text{notes al. } \textit{numal} & \rightarrow & \end{array}$$

De moment farem que el programa tregui només una llista amb el cognom i les notes de cada alumne. Un primer esquema del programa en pseudocodi podria ser:[27]

Programa Llista de notes
| (declaració del vector *cognom* i de la matriu *notes*)
| llegir el nombre d'alumnes (*numal*)
| llegir els cognoms i les notes dels alumnes
| escriure els cognoms i les notes dels alumnes
fi programa

L'acció "llegir els cognoms i les notes dels alumnes" requerirà un algorisme repetitiu estès a tots el alumnes:

acció llegir els cognoms i les notes dels alumnes
| per a i des de 1 fins a *numal* fer
| | llegir el cognom de l'alumne i
| | llegir les notes de l'alumne i
| fi per a
fi acció

L'acció "llegir les notes de l'alumne i" implica, al seu torn, una repetició estesa a les 10 qüestions:

acció llegir les notes de l'alumne i
| per a j des de 1 fins a 10 fer
| | llegir la nota de la qüestió j de l'alumne i ($notes(i, j)$)
| fi per a
fi acció

No detallarem l'acció "escriure els cognoms i les notes dels alumnes", ja que té una estructura anàloga a la de l'acció "llegir els cognoms i les notes dels alumnes".

La transcripció al Fortran de la part del programa que llegeix les dades seria:

```
character*12 cognom
dimension cognom(150), notes(150,10)
write (6,*) "Entreu el nombre d'alumnes examinats"
```

[27] Tot i que les declaracions no són accions, les inclourem en el pseudocodi (entre parèntesi) per tal que el lector no se n'oblidi en fer la transcripció al llenguatge Fortran.

```
      write (6,*) "(no pot ser superior a 150)"
      read (5,*) numal
c Ara llegirem els cognoms dels alumnes i les seves notes
      do i=1,numal
        write (6,*) "Entreu el cognom de l'alumne",i," i premeu Retorn"
        read (5,*) cognom(i)
          do j=1,10
            write (6,*) "Entreu la nota",j," de l'alumne",i,
     $                  " i premeu Retorn"
            read (5,*) notes(i,j)
          end do
      end do
```

Observeu que hem inclòs instruccions `write` per indicar a l'usuari el número d'ordre de l'alumne i la nota que toca llegir a cada iteració, així com una línia de comentari que conté informació dirigida al programador. Una taula amb els valors que va prenent cada índex facilita el seguiment de la lectura de cognoms i notes:

i	**informació llegida**	j	**informació llegida**
1	`cognom(1)`	1	`notes(1,1)`
		2	`notes(1,2)`
		...	...
		10	`notes(1,10)`
2	`cognom(2)`	1	`notes(2,1)`
		2	`notes(2,2)`
		...	...
		10	`notes(2,10)`
...	...	...	...
`numal`	`cognom(numal)`	1	`notes(numal,1)`
		...	...
		10	`notes(numal,10)`

Si volem que l'usuari només hagi de prémer Retorn una sola vegada per cada alumne, haurem d'ajuntar la lectura de cada cognom i les notes corresponents en una mateixa instrucció `read`, tot canviant el `do` més intern per un d'implícit:

```
      do i=1,numal
        write (6,*) "Entreu el cognom i les notes de l'alumne",
     $              i," i premeu Retorn"
        read (5,*) cognom(i), (notes(i,j), j=1,10)
      end do
```

Per escriure la llista amb el cognom i les notes de cada alumne en una mateixa línia haurem d'utilitzar, de nou, un *do* implícit:

```
      write (6,*) "Llista de notes de cada questio:"
      do i=1,numal
        write (6,*) cognom(i), (notes(i,j), j=1,10)
      end do
```

La llista que obtindrem tindrà, més o menys, l'aspecte següent:

```
Casals      4 8 6 9 3 5 1 3 4 7
Coll        6 8 7 3 5 9 7 4 5 5
...
Prat        6 8 7 3 4 2 4 8 7 9
```

Exercici 2.75
Indiqueu la quantitat de memòria RAM que ocuparan les variables declarades al programa anterior. *Solució*: 7800 bytes.

Exercici 2.76
Feu un programa que llegeixi el cognom i les qualificacions d'un examen amb diverses qüestions (cada una puntuada sobre 10), essent el nombre d'alumnes i el de qüestions dades a introduir per l'usuari (amb valors màxims de 150 i 30, respectivament), calculi la nota global (sobre 10) de cada alumne, tot guardant aquests resultats en un vector, i escrigui una llista amb el cognom de cada alumne, les notes que ha tret a cada qüestió i la mitjana d'aquestes (nota global), tot a la mateixa línia. *Observació*: Tingueu en compte que, fins i tot si les notes de les qüestions són enteres, la nota global pot no ser-ho.

Exercici 2.77
Afegiu al programa de l'exercici anterior el càlcul de la mitjana i la desviació típica de les notes globals (vegeu l'exercici 2.63).

Exercici 2.78
Feu un programa que llegeixi les qualificacions d'un examen amb diverses qüestions (cada una puntuada sobre 10), essent el nombre d'alumnes i el de qüestions dades a introduir per l'usuari (amb valors màxims de 150 i 30, respectivament), calculi la mitjana de les qualificacions de cada qüestió —tot guardant aquests resultats en un vector— i escrigui una llista amb el número de la qüestió i la seva nota mitjana.

Exercici 2.79
Feu un programa que demani a l'usuari el nombre de files (n) i els elements del triangle inferior (inclosa la diagonal principal) d'una matriu quadrada A; és a dir, els elements A_{ij} amb $i \geq j$. A partir d'aquests, el programa ha d'omplir la resta de la matriu de manera que aquesta sigui simètrica ($A_{ij} = A_{ji}$ per a i i j des d'1 fins a n) i ha d'escriure la matriu resultant (completa) per files.

Exercici 2.80
Modifiqueu el programa de l'exercici anterior perquè llegeixi els elements del triangle superior en comptes de l'inferior; és a dir, els elements A_{ij} amb $i \leq j$.

Exercici 2.81
Feu un programa que llegeixi el nombre de files (n) i els elements d'una matriu quadrada A, transformi aquesta en la seva transposada A^t (vegeu l'exercici 2.71) sense utilitzar cap altra matriu, i escrigui la matriu resultant per files. *Suggeriment*: Tingueu en compte que el programa només ha d'intercanviar els valors de les parelles $A_{12} \leftrightarrow A_{21}$, $A_{13} \leftrightarrow A_{31}$, $\ldots$, $A_{1n} \leftrightarrow A_{n1}$, $A_{23} \leftrightarrow A_{32}$, $\ldots$, $A_{n-1,n} \leftrightarrow A_{n,n-1}$.

Exercici 2.82
Feu un programa que llegeixi el nombre de files i el de columnes i els elements d'una matriu A, canviï el signe dels elements de les columnes amb índex senar, i escrigui la matriu resultant per files.

Exercici 2.83
Feu un programa que llegeixi el nombre de files (n) —que ha de ser senar— i els elements d'una matriu quadrada, sumi 1 a l'element que ocupa la posició central i resti 1 als altres elements, i escrigui la matriu resultant.

Exercici 2.84
Feu un programa que llegeixi els elements (enters) d'una matriu MAT de m files i n columnes, essent m i n dades a introduir per l'usuari, escrigui la matriu i ens indiqui quants n'hi ha de nuls a cada fila i quants n'hi ha a cada columna.

2.12 Entrada i sortida amb fitxers

Quan un programa ha d'escriure molts resultats és poc pràctic mirar-los al monitor a mesura que es van escrivint. A més, pot ser que vulguem utilitzar els resultats com a dades per a un altre programa o per a qualsevol altra aplicació; per exemple, un editor de textos per elaborar un informe, un full de càlcul o un programa gràfic per obtenir representacions gràfiques dels resultats, etc. Tot i que, si no hi ha gaires resultats, els podríem copiar des del monitor a un fitxer (a mà o mitjançant l'eina per copiar de la GUI), és més còmode i segur que ho faci el mateix programa que els ha calculat. Anàlogament, si un programa necessita moltes dades interessarà, per no haver de teclejar-les cada vegada que l'executem, que les llegeixi d'un fitxer.

D'altra banda, quan diversos usuaris treballen amb un mateix ordinador executant programes que requereixen molt temps de càlcul convé utilitzar un sistema de *cues de batch*: els usuaris envien els seus càlculs a alguna de les cues prèviament definides i l'ordinador va executant-los segons les seves disponibilitats de temps de procés, memòria RAM, espai en disc, etc. Com que l'usuari no sap quan s'executaran els seus programes, no pot estar pendent d'introduir-hi les dades o veure els resultats (pot ser que s'executin a les 5 de la matinada!), i cal que els programes agafin les dades de fitxers creats prèviament i escriguin els resultats en fitxers de sortida. L'usuari pot consultar l'estat de la cua cada cert temps o es pot fer que l'ordinador li enviï un correu electrònic quan s'acabi un càlcul seu. Llavors, aquest pot connectar-se a l'ordinador quan li vagi bé per obrir el corresponent fitxer de sortida i consultar-ne els resultats. Si no s'utilitzen cues de *batch*, és a dir, si es vol que un programa s'executi immediatament en donar l'ordre corresponent, es diu que s'està treballant *en interactiu*. Els exemples presentats en aquest text estan concebuts per a una persona que estigui treballant d'aquesta manera.

El GNU/Linux, com la major part dels sistemes operatius, permet dirigir la sortida d'un programa executable `prog` a un fitxer `prog.res` sense haver de modificar el programa, executant-lo amb la instrucció següent:

```
./prog > prog.res
```

Anàlogament, es pot fer que el programa llegeixi les dades d'un fitxer `prog.dad` —prèviament creat— en comptes del teclat, executant-lo amb la instrucció:

```
./prog < prog.dad
```

i, si es vol que s'utilitzin fitxers tant en l'entrada com en la sortida,

```
./prog < prog.dad > prog.res
```

El fitxer `prog.dad` haurà de contenir les mateixes dades que entraríem des del teclat en una execució normal, i tindrà una línia per cada instrucció `read` del programa. Si no s'especifica cap camí en la

designació dels fitxers d'entrada i/o sortida s'entén que aquests s'ubiquen en el directori de treball.

Exercici 2.85
Executeu el programa de l'exercici 2.76 (sense modificar-lo) de manera que llegeixi les dades d'un fitxer creat prèviament.

Exercici 2.86
Executeu el programa de l'exercici 2.76 (sense modificar-lo) de manera que tot el que escrigui ho faci en un fitxer. Obriu després aquest fitxer amb un editor de textos (pot ser el mateix que feu servir per escriure els programes o qualsevol altre) per comprovar que conté la informació que treu el programa.

El procediment que hem descrit per escriure en un fitxer la sortida d'un programa té l'inconvenient que l'usuari no veu cap indicació en el monitor sobre la marxa de l'execució. Així, si el programa presentava missatges per orientar l'usuari sobre les dades que ha d'entrar, el redireccionament de la sortida cap a un fitxer fa que aquests missatges no apareguin al monitor; i si redireccionem només l'entrada de dades veurem missatges que no tenen sentit, ja que no s'haurà de teclejar cap dada. Una alternativa més versàtil consisteix a definir els fitxers d'entrada i/o sortida dins del programa Fortran. Això es pot fer afegint, abans del `read` o `write`, la instrucció `open`:

```
open (n, file="nom")
```

on n és un nombre enter (normalment ha de ser < 100) que identificarà el fitxer a la resta del programa i *nom* és el nom que té el fitxer de dades, si es tracta d'un d'entrada, o el nom que es vol donar al de resultats, si es tracta d'un fitxer de sortida. Si el de sortida ja existeix en executar-se el programa la instrucció `open` esborra el seu contingut[28] i, si no existeix, el crea. Per llegir dades del fitxer utilitzarem la instrucció

```
read (n,*) var1, var2 ...
```

i, per escriure-hi resultats,

```
write (n,*) varoco1, varoco2 ...
```

Com ja sabem, els nombres 5 i 6 estan reservats per identificar el teclat i el monitor. De vegades s'utilitzen altres nombres inferiors a 10 per identificar certs dispositius, per la qual cosa és recomanable usar nombres de dos dígits per identificar fitxers.

Per exemple, si estem fent un programa per sumar matrius i no volem haver d'escriure les matrius a sumar cada vegada que el provem, podem crear un fitxer i guardar-hi dues matrius, escrites de manera idèntica a com ho faríem en una lectura de dades des del teclat. El programa podria ser:

```
      dimension a(100,100), b(100,100), c(100,100)
      write (6,*) "Aquest programa suma dues matrius guardades al fitxer
     $ sumat.dad"
      open (10, file="sumat.dad")
c Ara llegirem els nombres de files i de columnes de les matrius
      read (10,*) nfil, ncol
c Ara llegirem la primera matriu
      do i=1, nfil
        read (10,*) (a(i,j), j=1,ncol)
      end do
c Ara llegirem la segona matriu
      do i=1, nfil
        read (10,*) (b(i,j), j=1,ncol)
      end do
```

[28] Si volem que no s'esborri el contingut del fitxer hem d'afegir l'opció `,access="append"` en el parèntesi de la instrucció `open`.

```
c Ara sumarem les matrius
      do i=1, nfil
        do j=1, ncol
          c(i,j) = a(i,j) + b(i,j)
        end do
      end do
c Ara escriurem la matriu suma al monitor
      write (6,*) "Matriu suma:"
      do i=1, nfil
        write (6,*) (c(i,j), j=1,ncol)
      end do
      end
```

Si creem un fitxer amb nom `sumat.dad` i el contingut següent:

```
 3  3
 1  2  3
 4  5  6
 7  8  9
10 10 10
20 20 20
30 30 30
```

i executem el programa, la informació que veuríem en el monitor és:

```
Aquest programa suma dues matrius guardades al fitxer sumat.dad
Matriu suma:
11 12 13
24 25 26
37 38 39
```

Si volguéssim que la matriu suma s'escrigués en un segon fitxer podríem executar el programa —que anomenarem `sumat`— amb la instrucció

```
./sumat > sumat.res
```

Al monitor no veuríem cap missatge, i al fitxer `sumat.res` es guardaria tota la informació que abans apareixia en el monitor. En canvi, si canviem les instruccions d'escriptura de la matriu per:

```
c Ara escriurem la matriu suma en el fitxer sumat.res
      write (6,*) "La matriu suma es guarda en el fitxer sumat.res"
      open (11, file="sumat.res")
      do i=1, nfil
        write (11,*) (c(i,j), j=1,ncol)
      end do
```

veuríem en el monitor els missatges

```
Aquest programa suma dues matrius guardades al fitxer sumat.dad
La matriu suma es guarda en el fitxer sumat.res
```

i només s'escriuria en el fitxer `sumat.res` la matriu suma.

Exercici 2.87
Modifiqueu el programa de l'exercici 2.76 (pàgina 63) perquè escrigui la llista de notes totals en un fitxer, tot mantenint les indicacions dirigides a l'usuari a través del monitor per indicar-li com s'han d'entrar les dades.

El nom del fitxer especificat en una instrucció `open` es pot indicar mitjançant una variable literal. Això permet fer que aquest nom es llegeixi durant l'execució del programa. Per exemple, un programa que contingui les instruccions següents:

```
character*20 fitxer
write (6,*) "Dona'm el nom del fitxer de dades (< 21 caracters)"
read (5,*) fitxer
open (50, file=fitxer)
read (50,*) ...
```

permetrà a l'usuari escollir el nom que vulgui per al fitxer de dades. Si l'anomena `calcul.dades` haurà d'entrar aquest nom quan el programa ho demani.

Exercici 2.88
Feu un programa que demani a l'usuari els nombres de files i de columnes i els elements de dues matrius de las mateixes dimensions, en calculi la suma i la diferència i escrigui, per files, la suma en un fitxer i la diferència en un altre.

Exercici 2.89
a) Modifiqueu el programa de l'exercici 2.76 (pàgina 63) perquè permeti a l'usuari escollir entre introduir les dades des del teclat o llegir-les d'un fitxer creat prèviament, tot donant-li la informació adient per a cada cas. En el segon cas, el programa haurà de demanar a l'usuari el nom del fitxer.
b) Afegiu al programa anterior l'opció de treure els resultats pel monitor o escriure'ls en un fitxer.

2.13 Subprogrames

Suposem que hem desenvolupat un programa per calcular sumes de matrius (exercici 2.69, pàgina 60) i en volem fer un de nou que implica diverses sumes de diferents matrius. Podríem copiar el primer programa en cada punt del segon en què s'hagi de fer una suma, i adaptar a cada cas els noms de les variables i les dimensions de les matrius a sumar. Això, però, resultaria enutjós, conduiria a un programa força llarg i seria fàcil cometre errors en fer les adaptacions. Una opció més pràctica consisteix a transformar el programa per sumar matrius en un *subprograma*, i en el nou programa —que anomenarem *programa principal*— incloure una instrucció de *crida* al subprograma cada vegada que calgui fer aquella operació. A cada instrucció de crida s'haurà d'indicar quines són les matrius que es volen sumar.

En general, un programa principal pot anar acompanyat de diversos subprogrames que es poden incloure en el mateix fitxer (després de la instrucció `end` del principal), encara que també poden estar en fitxers diferents, compilar-se per separat i muntar-se després juntament amb al programa principal per obtenir l'executable. Si un mateix fitxer conté diversos subprogrames, l'ordre d'aquests és irrellevant. Tots els subprogrames que utilitzi un programa principal han de tenir noms diferents, els quals han d'obeir les mateixes regles que els de les variables (vegeu l'apartat 2.3.4). Qualsevol subprograma pot cridar un altre subprograma mentre no es cridi a ell mateix ni directament ni indirectament (és a dir, a través d'un altre subprograma).[29]

El Fortran proporciona dos tipus de subprogrames: les subrutines i les funcions externes.

2.13.1 Subrutines

Una *subrutina* és un subprograma que comença amb la instrucció de declaració `subroutine`:

```
subroutine nom (arg1,arg2, ..., argn)
```

on *nom* és el nom que es vulgui donar a la subrutina i *arg1*, *arg2*, ..., *argn* és una llista de variables que anomenarem els *arguments* de la subrutina. L'estructura de la resta del subprograma és anàloga a la d'un programa principal: resta d'instruccions de declaració, instruccions executables i instrucció `end`.[30] La crida a una subrutina es fa mitjançant una instrucció `call`:

[29] Altres llenguatges, així com les versions més recents del Fortran, inclouen la possibilitat que un subprograma es pugui cridar a ell mateix (*recursivitat*).

[30] El compilador identifica com a programa principal les instruccions que no formen part de cap subprograma, de manera que no cal cap instrucció de declaració per identificar-lo. No obstant això, es pot incloure com a primera

```
call nom (arg1,arg2, ..., argn)
```

on els arguments[31] han de ser dades —variables o constants[32]— del mateix tipus i estar en el mateix ordre que les variables de la instrucció `subroutine`,[33] encara que poden tenir noms diferents.

Com de costum, ho il·lustrarem amb un exemple senzill: un programa que llegeixi dos nombres reals i els passi a una subrutina que en calculi la mitjana geomètrica i l'escrigui. El subprograma, que anomenarem `mitgeo`, comprovarà que els nombres a amitjanar siguin del mateix signe. Si no ho són, la mitjana geomètrica no es pot calcular, i la subrutina emetrà un missatge indicant-ho. El programa complet podria ser aquest:

```
c Programa principal
      write (6,*) "Escriviu els nombres que voleu amitjanar"
      read (5,*) a, b
      call mitgeo(a, b)
      end
c Ara ve la subrutina mitgeo
      subroutine mitgeo(a, b)
      if (a*b .ge. 0.) then
        abmgeo = sqrt(a*b)
        write (6,*) "La seva mitjana geometrica es",abmgeo
      else
        write (6,*) "No s'ha pogut calcular la mitjana"
      end if
      end
```

Quan s'executa la instrucció de crida a la subrutina `mitgeo`:

```
call mitgeo(a, b)
```

el programa principal passa a la subrutina les variables `a` i `b` i comença l'execució del subprograma. Aquest fa la comprovació abans esmentada i n'escriu el resultat. En acabar-se l'execució del subprograma, continua la del programa principal a partir de la instrucció següent a la crida; en aquest cas, com que no n'hi ha cap més d'executable, s'aturarà.

En aquest exemple la utilització d'una subrutina per calcular la mitjana no presenta avantatges evidents sobre l'opció de fer el càlcul en el programa principal. Aquests avantatges s'apreciaran millor en el programa següent, on s'avaluaran diverses mitjanes geomètriques amb la mateixa subrutina. El subprograma retornarà al programa principal el resultat del càlcul, o bé un valor absurd (per exemple, -1) si no s'ha pogut obtenir la mitjana. Aquesta manera de procedir —escriure els resultats en el programa principal— és també més adient si es vol reutilitzar la subrutina en altres programes —o, fins i tot, per altres programadors— tot adaptant la presentació dels resultats a cada cas sense haver de modificar el subprograma. Llavors convé incloure al principi d'aquest subprograma un breu comentari explicatiu del que fa i del significat dels arguments, com s'ha fet en l'exemple:

```
c Programa principal
      write (6,*) "Escriviu els nombres que voleu amitjanar"
```

sentència del programa principal una instrucció `program`, la qual, a més, permet assignar-li un nom. Per exemple:

```
program prog1
...
end
subroutine subr1
...
end
```

[31] De vegades es fa servir el terme *paràmetres* per referir-se a les variables de la instrucció `subroutine` i es reserva la paraula *arguments* per a les variables o constants de la instrucció de crida. El mateix s'aplica a les funcions externes.

[32] Certs compiladors admeten també expressions amb variables i constants, les quals s'avaluen en executar-se la crida i es passa a la subrutina el valor resultant. És preferible, però, assignar prèviament l'expressió a una variable i passar aquesta com a argument de la crida.

[33] Si el subprograma es compila independentment del programa que fa la crida, el compilador no pot verificar aquesta correspondència, i el programador haurà de tenir cura d'assegurar-la.

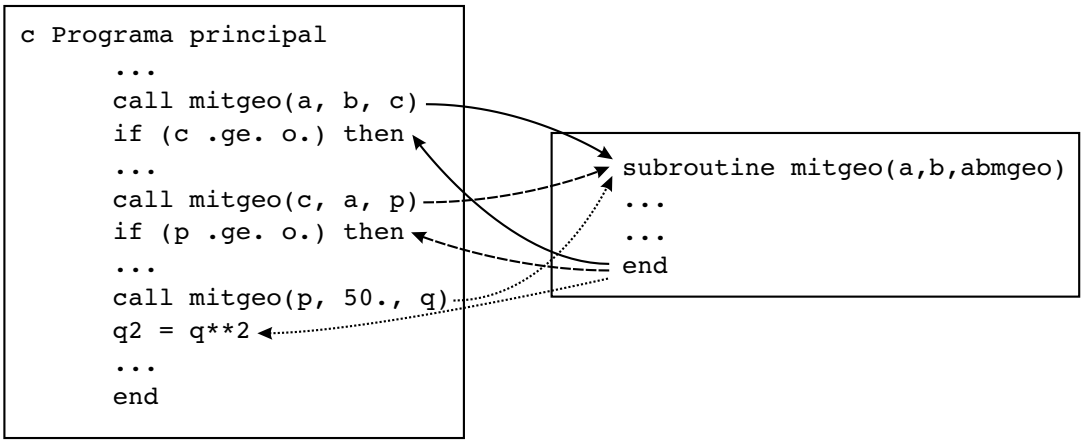

Figura 2.2: Flux de control d'un programa en diferents crides a una mateixa subrutina.

```
      read (5,*) a, b
      call mitgeo(a, b, c)
      if (c .ge. 0.) then
        write (6,*) "La seva mitjana geometrica es",c
      else
        write (6,*) "No s'ha pogut calcular la mitjana"
        stop
      end if
      write (6,*) "Ara calcularem la mitjana entre ",c," i ",a
      call mitgeo(c, a, p)
      if (p .ge. 0.) then
        write (6,*) p
      else
        write (6,*) "No s'ha pogut calcular la mitjana"
        stop
      end if
      write (6,*) "i ara el quadrat de la mitjana entre ",p," i 50"
      call mitgeo(p, 50., q)
      q2 = q**2
      write (6,*) q2
      end
      subroutine mitgeo(a, b, abmgeo)
c     Aquesta subrutina calcula la mitjana geometrica dels
c     arguments reals a i b i la retorna en l'argument real abmgeo
c     (si a i b son de signe diferent retorna un -1. com a codi d'error)
      if (a*b .ge. 0.) then
        abmgeo = sqrt(a*b)
      else
        abmgeo = -1.
      end if
      end
```

Quan s'executa la primera instrucció de crida a la subrutina `mitgeo`:

```
      call mitgeo(a, b, c)
```

passa a executar-se la subrutina i, en acabar aquesta, continua l'execució del programa principal a partir de la instrucció següent a la crida (línies contínues de la figura 2.2). En aquesta primera execució de la subrutina, els arguments `a`, `b` i `abmgeo` representen les mateixes posicions de la memòria RAM que les variables `a`, `b` i `c`, respectivament, del programa principal; per tant, la instrucció

```
abmgeo = sqrt(a*b)
```

de la subrutina calcularà la mitjana geomètrica dels valors llegits per a `a` i `b` al programa principal i guardarà el resultat a la variable `c` d'aquest, que correspon a l'`abmgeo` de la subrutina. En la segona crida a la subrutina el programa principal li passa com a arguments les variables `c`, `a` i `p`:

```
call mitgeo(c, a, p)
```

és a dir, ara seran aquestes les variables del programa principal que es corresponen amb els arguments `a`, `b` i `abmgeo` de la subrutina (línies de traços de la figura 2.2). En conseqüència, aquesta calcularà la mitjana de les variables `c` i `a` del programa principal i posarà el resultat en la variable `p` del mateix programa. La tercera crida

```
call mitgeo(p, 50., q)
```

calcularà la mitjana de `p` i 50 i la guardarà en `q` (línies de punts de la figura 2.2). Observem que, en aquesta última crida, és important posar el punt a la constant 50 per tal que aquesta es codifiqui com a real i el seu tipus es correspongui amb el de la variable `b` de la subrutina. Com que els nombres a amitjanar són, en aquest cas, ambdós positius, no cal preveure el cas que la mitjana no es pugui calcular.

Els dos primers arguments de la subrutina `mitgeo` passen informació del programa principal a la subrutina, i direm que són *d'entrada*; el tercer, en canvi, retorna informació al programa principal i diem que és *de sortida*. Una subrutina pot tenir tants arguments d'entrada o de sortida com es vulgui, i també n'hi poden haver que siguin *d'entrada i sortida* a la vegada; és a dir, que passin dades a la subrutina i aquesta hi retorni resultats. És evident que, en una instrucció de crida, no podrem passar una constant com a argument de sortida (donaria un error d'execució).

Exercici 2.90
Feu un programa que llegeixi un nombre real i el passi a una subrutina que determini el seu valor absolut (sense utilitzar la funció incorporada `abs(x)`) i el retorni al programa principal. Aquest ha d'escriure el valor absolut.

Exercici 2.91
a) Feu un programa que llegeixi el nombre de mols, la temperatura absoluta en kelvins i la pressió en bars d'un gas ideal i passi aquestes dades a una subrutina que calculi el volum que ocupa el gas en litres i retorni el valor obtingut al programa principal. Aquest haurà d'escriure el volum que ocupa el gas (recordeu que $R = 0{,}08314472$ bar l K^{-1} mol^{-1}). Si s'entren valors nuls o negatius per a la pressió i/o la temperatura i/o el nombre de mols, la subrutina haurà de tornar un codi d'error al programa principal i aquest haurà d'aturar l'execució després d'emetre un missatge explicatiu. El codi d'error pot ser, per exemple, una variable entera que prengui el valor 0 si les dades són acceptables i 1 si no ho són.
b) Modifiqueu el programa principal perquè faci tres crides a la subrutina; en la primera, li passarà la temperatura introduïda per l'usuari; en la segona, la temperatura augmentada en 10 kelvins, i en la tercera, la temperatura disminuïda en 10 kelvins. A continuació de cada crida el programa principal ha d'escriure la temperatura absoluta i el volum corresponent.

Exercici 2.92
Feu un programa que llegeixi dos nombres enters i cridi a una subrutina anomenada `quoent` que, sense utilitzar la funció incorporada `mod`, calculi la part entera del quocient i el residu que resulten de dividir aquelles dades i retorni els valors obtinguts al programa principal (vegeu l'exercici 2.7, pàgina 32). Aquest ha d'escriure els resultats.

Les úniques dades que comparteix la subrutina amb el programa principal són els arguments; és a dir, qualsevol variable o constant del subprograma que no aparegui a la instrucció `subroutine` representa una zona de la memòria RAM diferent de les de qualsevol variable o constant del programa principal —independentment dels noms que tinguin les unes i les altres— i, viceversa, les dades del

programa principal que no apareixen en una instrucció de crida són independents de les dades de la subrutina, encara que tinguin el mateix nom. Es diu que les variables o constants que no són arguments són *locals*.[34] Així, el programa

```
a = 1.
b = 2.
c = 3.
d = 4.
e = 5.
f = 6.
call subr1(a, b)
write (6,*) a, b, c, d, e, f
end
subroutine subr1(e, f)
a = 7.
b = 8.
c = 9.
d = 10.
e = (a+b)*e
f = (c+d)*f
end
```

escriurà:

```
15.  38.  3.  4.  5.  6.
```

Observeu que els dos arguments de la subrutina d'aquest programa són d'entrada i sortida a la vegada.

Exercici 2.93
Indiqueu els valors que escriurà el programa següent:

```
a = 1.
b = 2.
c = 3.
call me(a, b, c)
call me(c, b, a)
write (6,*) a, b, c
end
subroutine me(a, b, c)
x = a
a = c
c = b
b = x
end
```

Solució: 1. 2. 3.

Normalment, l'execució d'una subrutina s'acaba amb l'última instrucció executable; no obstant això, es pot fer que el control de l'execució torni abans al programa principal incloent-hi una instrucció `return`. Per exemple, la subrutina per calcular mitjanes geomètriques del programa de la pàgina 68 es podria haver programat de la manera següent:

```
subroutine mitgeo(a, b, abmgeo)
if (a*b .lt. 0.) then
  abmgeo = -1.
  return
end if
```

[34] Es poden definir variables comuns a diferents (sub)programes sense que apareguin en instruccions `call` o `subroutine` mitjançant la instrucció `common`, que serà considerada en la secció 3.6.

```
abmgeo = sqrt(a*b)
end
```

Si s'hagués posat una instrucció `stop` en comptes de la instrucció `return`, en executar-se s'aturaria el programa, de manera que el control de l'execució no retornaria al programa principal.

Exercici 2.94
a) Feu un programa que demani un nombre natural i el passi a una subrutina que determini si és primer o no i escrigui aquesta informació (vegeu l'exercici 2.33, pàgina 45).
b) Modifiqueu el programa anterior perquè, si l'usuari entra un nombre negatiu, la subrutina emeti un missatge d'error i aturi l'execució del programa.
c) Modifiqueu el programa de l'apartat *b)* per tal que, si l'usuari entra un nombre negatiu, la subrutina torni anticipadament el control de l'execució al programa principal i sigui aquest el que s'aturi després d'emetre el missatge explicatiu. *Suggeriment*: Podeu fer que la subrutina retorni al programa principal un codi d'error, tal com s'explica en l'exercici 2.91.

Exercici 2.95
Feu un programa que demani a l'usuari un nombre natural n i escrigui els nombres primers inferiors a n. *Suggeriment*: Podeu aprofitar l'algorisme suggerit en l'exercici 2.33, tot convertint el programa que vau fer llavors en una subrutina.

Exercici 2.96
Modifiqueu el programa anterior perquè calculi i escrigui la suma dels nombres primers més petits o iguals que n. *Comprovació*: Per a $n = 12$ ha de donar 29 $(= 1 + 2 + 3 + 5 + 7 + 11)$.

Quan diem que els arguments de la instrucció de crida a una subrutina han de ser del mateix tipus que els de la primera instrucció d'aquesta, ens referim no solament al tipus de dades (enter, real, etc.) sinó també al caràcter escalar (no indexat) o indexat de les variables. Això implica que, si un argument de la crida és un vector o una matriu, haurà d'estar declarat com a tal també en la subrutina i amb les mateixes dimensions. Per exemple:

```
dimension a(50,100), b(75)
integer c(10,10)
character*15 titol
...
call sub(a, b, c, titol)
...
end
subroutine sub(x, y, mat, tit)
dimension x(50,100), y(75), mat(10,10)
character*15 tit
...
end
```

Exercici 2.97
Feu un programa que llegeixi un vector de 10 components i cridi a una subrutina que canviï el signe de tots ells. El programa principal ha d'escriure el vector canviat de signe.

Exercici 2.98
Modifiqueu el programa anterior perquè serveixi per a vectors de dimensió qualsevol (fins a 100). El programa principal ha de llegir el nombre de components del vector i passar-lo a la subrutina.

Observem que quan es passa una variable indexada completa a una subrutina s'escriu el nom d'aquesta sense parèntesis, tant en la llista d'arguments de la crida com en la de la instrucció `subroutine`. En canvi, si volguéssim passar un sol component de la variable indexada s'haurien d'utilitzar parèntesis per especificar-ho. Per exemple, en el programa:

```
dimension a(50,100), b(75)
...
call sub(a(1,3), b, d)
...
end
subroutine sub(x, y, d)
dimension y(75)
...
end
```

el primer argument `x` de la subrutina és escalar i, d'acord amb això, la instrucció de crida li passa un sol element de la matriu `a`: el `a(1,3)`. El següent argument `y` és un vector que es correspon amb el vector `b` de la instrucció de crida, i l'últim argument és una variable no indexada que té el mateix nom en el programa principal i en la subrutina. L'única matriu declarada en el programa principal no es passa como a tal a la subrutina, per la qual cosa no cal declarar-la com a matriu en aquest subprograma.

Com a exemple d'utilització de subrutines amb paràmetres que siguin matrius, presentarem la versió següent del programa proposat en l'exercici 2.69 a) (pàgina 60), que calcula la suma de dues matrius de 4 files i 3 columnes:

```
dimension a(4,3), b(4,3), c(4,3)
write (6,*) "Dona'm els elements de la primera matriu per files"
call lecmat(a)
write (6,*) "Dona'm els elements de la segona matriu per files"
call lecmat(b)
call sumat(a, b, c)
write (6,*) "La matriu suma de"
call escmat(a)
write (6,*) "i"
call escmat(b)
write (6,*) "es"
call escmat(c)
end
subroutine lecmat(a)
dimension a(4,3)
do i=1,4
  read (5,*) (a(i,j), j=1,3)
end do
end
subroutine sumat(a,b,c)
dimension a(4,3), b(4,3), c(4,3)
do i=1,4
  do j=1,3
    c(i,j) = a(i,j) + b(i,j)
  end do
end do
end
subroutine escmat(a)
dimension a(4,3)
do i=1,4
  write (6,*) (a(i,j), j=1,3)
end do
end
```

Exercici 2.99
Modifiqueu el programa de l'exemple anterior perquè serveixi per a matrius de dimensions qualssevol (fins a 100×100). El programa principal ha de llegir els nombres de files i de columnes de les matrius i passar aquesta informació a cada subrutina.

Exercici 2.100
Feu un programa que llegeixi el nombre de components d'un vector (amb un màxim de 100) i els valors d'aquests, i passi aquestes dades a una subrutina que haurà d'invertir l'ordre dels components (el primer amb l'últim, el segon amb el penúltim, etc.) sense utilitzar cap altre vector. La subrutina ha de retornar el vector reordenat al programa principal, i aquest l'ha d'escriure. *Suggeriment*: Penseu primer quin és l'algorisme per a un nombre parell de components i generalitzeu-lo després per a un nombre qualsevol.

Observem que, quan fem un primer esquema en pseudocodi d'un programa, les accions més complicades es poden desenvolupar com a subrutines, de manera que el programa principal sigui una transcripció d'aquell primer esquema. Per exemple, la transcripció al llenguatge Fortran de l'algorisme per resoldre equacions de segon grau presentat en la pàgina 39 podria ser:

```
c Programa principal
      write (6,*) "Dona'm els coeficients a, b i c de l'equacio"
      read (5,*) a, b, c
      if (a .eq. 0.) then
        call eq1(b,c)
      else
        call eq2(a, b, c)
      end if
      end
```

A continuació vindrien les subrutines `eq1` i `eq2` per resoldre equacions de primer i segon grau, respectivament, i escriure'n les solucions.

Exercici 2.101
Completeu amb les subrutines adients el programa principal presentat en l'exemple anterior (vegeu l'exercici 2.20, pàgina 40).

Exercici 2.102
Feu un programa que llegeixi el nombre de components de dos vectors de la mateixa dimensió i els valors dels components, cridi dues vegades a una mateixa subrutina per calcular el mòdul de cada vector, cridi a una altra subrutina per calcular la diferència entre els vectors, i torni a cridar a la primera subrutina per calcular el mòdul d'aquesta diferència; és a dir, la distància entre els vectors llegits al començament. El programa principal ha d'escriure la suma dels mòduls dels vectors llegits i la distància entre ells. Utilitzeu el programa per comprovar que la distància entre els vectors és més petita o igual que la suma dels seus mòduls. *Comprovació*: La distància entre els vectors $(2, -1, 5)$ i $(1, 3, -2)$ és 8,12404, i la suma dels seus mòduls és 9,21888.

2.13.2 Funcions externes

Una funció externa[35] és un subprograma que comença amb la instrucció de declaració `function`:

[35] Aquests tipus de subprogrames s'anomenen funcions "externes" per distingir-los d'un altre tipus de funció que admet el Fortran: les funcions definides mitjançant una instrucció ubicada al final de les de declaració. Aquestes són "internes" en el sentit que només s'hi pot accedir des del (sub)programa on estan definides. En aquest text no utilitzarem funcions internes, les quals sempre poden ser substituïdes per funcions externes.

```
      function nom (arg1,arg2, ..., argn)
```

on *nom* és el nom de la funció i *arg1*, *arg2*, ..., *argn* és una llista de variables que actuen com a arguments d'entrada; és a dir, permeten transferir informació del programa que fa la crida a la funció externa. L'estructura de la resta del subprograma es anàloga a la d'un programa principal (resta d'instruccions de declaració, instruccions executables i instrucció `end`). Entre les instruccions executables ha d'haver una assignació a una variable escalar que tingui el mateix nom que la funció, i aquesta és l'únic valor que retorna el subprograma.[36] Per cridar una funció externa només cal posar el seu nom en qualsevol expressió seguit dels arguments —entre parèntesis— per als que vulguem avaluar-la. Aquests han de ser dades —variables, constants o expressions— del mateix tipus i estar en el mateix ordre que les variables de la instrucció `function`, encara que poden tenir noms diferents. Les funcions externes seran, doncs, l'elecció més adequada per codificar funcions matemàtiques d'una o més variables ($f(x_1, x_2, \ldots, x_n)$), ja que per a cada conjunt de valors d'aquestes la funció pren un únic valor.

Vegem com es pot implementar l'exemple de la pàgina 68 utilitzant una funció externa en lloc d'una subrutina per calcular la mitjana geomètrica:

```
c Programa principal
      write (6,*) "Escriviu els nombres que voleu amitjanar"
      read (5,*) a, b
      c = geom(a, b)
      if (c .ge. 0.) then
        write (6,*) "La seva mitjana geometrica es",c
      else
        write (6,*) "No s'ha pogut calcular la mitjana"
        stop
      end if
      write (6,*) "Ara calcularem la mitjana entre ",c,"i ",a
      p = geom(c, a)
      if (p .ge. 0.) then
        write (6,*) p
      else
        write (6,*) "No s'ha pogut calcular la mitjana"
        stop
      end if
      write (6,*) "i ara el quadrat de la mitjana entre ",p,"i 50"
      q2 = geom(p, 50.)**2
      write (6,*) q2
      end
      function geom(a, b)
c     Aquesta funcio externa calcula la mitjana geometrica
c     dels arguments reals a i b
c     (si a i b son de signe diferent retorna un -1. com a codi d'error)
      if (a*b .ge. 0.) then
        geom = sqrt(a*b)
      else
        geom = -1.
      end if
      end
```

Observem que la crida a una funció externa té la mateixa forma que la crida a una funció incorporada (`cos(x)`, `max(x1,x2,...,xn)`, etc.); de fet, aquestes són funcions externes que se subministren, ja compilades, juntament amb el compilador i s'incorporen a l'executable durant el muntatge (vegeu la secció 2.2).

[36] En realitat els compiladors solen permetre modificar els arguments de les funcions externes, de manera que aquests poden actuar també com a paràmetres de sortida. No obstant això, aquesta no és la manera habitual de treballar i recomanem utilitzar una subrutina quan s'hagi de retornar més d'una dada.

Si no s'ha declarat d'una altra manera, el valor que retorna una funció és enter o real segons quina sigui la primera lletra del seu nom: si aquesta està entre la 'i' i la 'n' la funció prendrà valors enters i si està entre la 'a' i la 'h' o entre la 'o' i la 'z' prendrà valors reals. Així, si en l'exemple anterior haguéssim anomenat la funció externa `mitgeo` en comptes de `geom`, aquesta retornaria la part entera de la mitjana. Per evitar-ho s'hauria de declarar `mitgeo` com a variable real tant en el programa principal com en el subprograma:

```
c Programa principal
      real mitgeo
      write (6,*) "Escriviu els nombres que voleu amitjanar"
      read (5,*) a, b
      c = mitgeo(a, b)
      ...
      end
      function mitgeo(a, b)
c     Aquesta funcio externa calcula la mitjana geometrica
c     dels arguments reals a i b
c     (si a i b son de signe diferent retorna un -1. com a codi d'error)
      real mitgeo
      if (a*b .ge. 0.) then
        mitgeo = sqrt(a*b)
      else
        mitgeo = -1.
      end if
      end
```

En el subprograma, la declaració del tipus de funció com a real, doble precisió, character*12, etc. es pot fer també en la mateixa instrucció `function`:

```
      real function mitgeo(a, b)
```

El fet que la crida a una funció es pugui incloure en una expressió matemàtica (o d'altre tipus) fa que, en certs casos, sigui més còmode utilitzar aquest tipus de subprogrames en comptes de les subrutines. Així, en el programa principal de l'exemple anterior la instrucció

```
      q2 = geom(p, 50.)**2
```

s'ha de substituir per dues instruccions si s'utilitza una subrutina en lloc d'una funció externa per calcular la mitjana (vegeu l'exemple de la pàgina 68):

```
      call mitgeo(p, 50., q)
      q2 = q**2
```

Exercici 2.103
Modifiqueu el programa de l'exercici 2.91 (pàgina 70) de manera que utilitzi un funció externa en comptes d'una subrutina per calcular el volum del gas ideal. Feu que la comprovació de les dades es faci en el programa principal, de manera que la funció només haurà de retornar el volum.

Exercici 2.104
a) Escriviu una funció externa entera amb nom `residu` que faci la mateixa tasca que la funció incorporada `mod` sense emprar aquesta.
b) Modifiqueu el programa de l'exercici 2.92 (pàgina 70) perquè la subrutina `quoent` utilitzi la funció `residu` per obtenir el residu de la divisió.

Els mateixos comentaris que hem fet per a les subrutines en relació amb el caràcter local de les variables que no són arguments, la declaració de variables indexades i la instrucció `return` s'apliquen

a les funcions.

Exercici 2.105
Escriviu una funció externa que calculi el producte escalar entre dos vectors de fins a 100 components (vegeu l'exercici 2.62, pàgina 56). Els arguments de la funció han de ser els vectors i el nombre de components d'aquests. Escriviu també un programa principal que llegeixi dos vectors, calculi el seu producte escalar fent servir aquesta funció i l'escrigui.

Exercici 2.106
Escriviu una funció externa que calculi la traça d'una matriu quadrada de fins a 100×100 elements (vegeu l'exercici 2.73, pàgina 61). Els arguments de la funció han de ser la matriu i el seu nombre de files. Feu també un programa principal que llegeixi una matriu quadrada, en calculi la traça fent servir aquesta funció i l'escrigui.

2.13.3 Avantatges dels subprogrames

A part de reduir l'extensió de programes en els quals es repeteixin algorismes similars, la utilització de subprogrames té molts altres avantatges:

- Facilita l'estructuració dels programes (vegeu la secció 2.7), ja que permet que el programa principal sigui una transcripció del primer nivell d'especificació del programa escrit amb pseudocodi, on cada acció complexa se substitueix per una crida a un subprograma que la desenvolupa. El mateix es pot fer amb els subprogrames que incloguin accions massa complexes.
- Facilita l'aprofitament del codi d'un programa per a d'altres programes que tinguin accions iguals o similars: si hem verificat que un subprograma funciona correctament, no haurem de tornar a preocupar-nos pel seu contingut; bastarà saber quins arguments s'han d'especificar en la instrucció de crida per poder utilitzar-lo en qualsevol altre programa.
- Facilita la millora dels programes: si trobem un algorisme més eficient per fer una acció, n'hi haurà prou a modificar el subprograma corresponent perquè la modificació afecti totes les crides que se'n facin.
- En programes molt llargs permet reduir notablement el temps necessari per compilar-los: els subprogrames que estan prou verificats es poden guardar compilats i, per obtenir l'executable, només caldrà compilar la part del programa en la qual estem treballant i muntar-la després amb la resta. Els subprogrames compilats es poden agrupar en *llibreries*. Per exemple, podríem construir una llibreria amb subprogrames per resoldre diferents operacions de càlcul matricial: lectura i escriptura de matrius, suma, producte i diagonalització de matrius, càlcul de la transposada, de la traça, etc.
- Hi ha moltes llibreries de subprogrames d'utilitat general —càlcul numèric, representacions gràfiques, etc.— disponibles com a programari lliure o comercial (vegeu-ne algunes a la bibliografia). El llibre *Numerical Recipes* proporciona rutines segures i eficients per resoldre molts problemes de càlcul numèric, i el *Numerical Algorithm Grup* (NAG) comercialitza una llibreria molt completa de càlcul numèric i representacions gràfiques. Les subrutines lliures *Basic Linear Algebra Subprograms* (BLAS) inclouen tota mena d'operacions amb vectors i matrius programades de manera molt eficient. La llibreria lliure *PGPLOT Graphics Subroutine Library* conté subprogrames per representar dades gràficament des d'un programa Fortran.

3 Informació addicional i altres exercicis

Ara que ja hem introduït les instruccions més importants del llenguatge Fortran i les hem il·lustrat amb aplicacions senzilles, estem en condicions d'abordar aplicacions una mica més complexes que ajudin a assimilar la matèria introduïda. Aprofitarem alguns d'aquests exercicis per incloure certa informació addicional relativa a instruccions prèviament considerades, i afegirem una breu explicació d'algunes instruccions que hem deixat de banda intencionadament. És el cas de la instrucció `format`, que permet personalitzar la presentació de resultats i la manera d'introduir dades; la instrucció `do` *amb etiqueta*, equivalent al `do` *amb índex* introduït en l'apartat 2.10.1; la instrucció `go to`, que, tot i ser en general desaconsellable, permet resoldre certes qüestions d'una manera molt senzilla, i les instruccions `data` i `common`.

3.1 Taules

Suposem que volem construir una taula amb els valors que pren una funció $f(x)$ per a valors de x distribuïts uniformement en un interval $[a, b]$. Aquesta taula es podria emprar, per exemple, per fer una representació gràfica posterior de la funció mitjançant una aplicació de dibuix adient. Si dividim l'interval $[a, b]$ en n subdivisions de longitud

$$h = \frac{b-a}{n}$$

el programa haurà d'avaluar i escriure els valors de x_i i de $f(x_i)$ per a $i = 0, 1, \ldots, n$, essent:

$$\begin{aligned} x_0 &= a \\ x_1 &= a + h \\ x_2 &= a + 2h \\ &\cdots \\ x_n &= a + nh = b \end{aligned}$$

és a dir, per als $n + 1$ valors:

$$x_i = a + ih \qquad \text{on} \quad i = 0, 1, \ldots, n$$

Exercici 3.1
Feu un programa que tabuli la funció $\sin(x)$ per a $n + 1$ valors equidistants de x entre $-\pi$ i π, on n —nombre de subdivisions— és una dada que s'ha de demanar a l'usuari (observeu que si doneu un valor parell a n es calcularà la funció per a $x = 0$). No cal que utilitzeu variables indexades per guardar els valors calculats.

Exercici 3.2
Feu un programa que tabuli la funció e^x per a $n+1$ valors equidistants de x en l'interval $[0,2]$, on n és una dada que haurà d'introduir l'usuari (no cal que utilitzeu variables indexades per guardar els valors calculats).

Exercici 3.3
Modifiqueu el programa de l'exercici 2.14 (pàgina 34) perquè vagi donant la concentració de N_2O a intervals regulars de temps des de l'inici de la reacció fins a un temps màxim en segons que haurà d'introduir l'usuari com a dada, juntament amb el nombre de subdivisions de l'interval de temps total.

Si es volen fer servir les dades d'una taula en una altra aplicació —per exemple, per representar-les gràficament— convé escriure-les en un fitxer amb un format adequat per poder llegir-les des de l'aplicació. Així, si escrivim cada parella de valors x_i i $f(x_i)$ en una línia del fitxer amb format lliure quedaran separats per espais en blanc i podrem llegir-los directament amb l'aplicació gràfica gnuplot, present en gairebé qualsevol distribució de GNU/Linux (també es pot descarregar, per a aquest o altres sistemes operatius, de la pàgina web *http://www.gnuplot.info*). Per exemple, per representar en un sistema d'eixos cartesians les parelles de valors, guardades en el fitxer `taula.txt`, només cal obrir l'aplicació gnuplot i teclejar:

```
plot "taula.txt"
```

També es pot emprar un full de càlcul, com ara el Calc del paquet de programari lliure OpenOffice (*http://ca.openoffice.org/*). En aquest cas convé escriure la taula en un fitxer amb extensió `csv` i format CSV; és a dir, amb una coma entre cada parella de valors x_i i $f(x_i)$ (vegeu *http://en.wikipedia.org/wiki/Comma-separated_values*). Si l'obrim amb l'aplicació Calc i acceptem les opcions suggerides quedaran les dades correctament disposades en columnes i a punt per fer una representació gràfica o qualsevol altre tractament.[1]

Exercici 3.4
a) Feu un programa que avaluï la funció $f(x) = \frac{\sin(x)}{x}$ per a n valors equidistants de x en l'interval $(0, 4\pi]$ i escrigui en el monitor una taula amb les n parelles de valors $x, f(x)$. Com que la funció presenta una indeterminació en $x = 0$, el primer punt que heu de considerar és $4\pi/n$.
b) Modifiqueu el programa anterior perquè la taula de valors s'escrigui en un fitxer amb format CSV, a més d'escriure's en el monitor.

Per tabular funcions de dues variables $f(x, y)$ caldran dos índexs, un per a la variable x i l'altre per a la variable y. Per exemple, podríem donar a la primera els $n+1$ valors

$$x_i = a + ih \qquad \text{on} \quad i = 0, 1, \ldots, n$$

i, per a cada un d'aquests, donar a la segona els $m+1$ valors (entre c i d)

$$y_j = c + jg \qquad \text{on} \quad j = 0, 1, \ldots, m \quad \text{i} \quad g = \frac{d-c}{m}$$

Per a cada una d'aquestes $(n+1)(m+1)$ parelles de valors, el programa haurà d'avaluar $f(x_i, y_j)$ i

[1] Per llegir-lo des de l'Excel heu d'obrir un document nou i seleccionar: Dades -> Obtenir dades externes -> Importar arxiu de text -> Activar tots els arxius -> Delimitats -> Següent -> Coma -> Següent -> Avançades -> Separador decimal: . -> Separador de milers: (res) -> OK -> Acabar.

escriure el resultat de l'avaluació precedit dels valors de x_i i de y_j.

Exercici 3.5
Feu un programa que tabuli la funció $\phi_{2p_x}(x,y) = \frac{1}{\sqrt{32\pi}} x e^{-\sqrt{x^2+y^2}/2}$ (que és l'orbital $2p_x$ d'un àtom d'hidrogen en el pla xy expressat en unitats adients) per a 21 valors equidistants de x entre –1 i 1, i 11 valors de y entre –0,5 i 0,5 (no cal que utilitzeu variables indexades). La taula següent indica com han d'anar canviant les variables més significatives del programa (els valors de la funció s'assignen a la variable `fi2px`):

`x`	`y`	`fi2px`
$-1,0$	$-0,5$	$-0,0570259$
	$-0,4$	$-0,0582070$
	...	...
	$0,5$	$-0,0570259$
$-0,9$	$-0,5$	$-0,0536446$
	$-0,4$	$-0,0548564$
	...	...
	$0,5$	$-0,0536446$
...	...	...
$1,0$	$-0,5$	$0,0570259$
	...	...
	$0,5$	$0,0570259$

Exercici 3.6
Tabuleu la funció $\phi_{2p_x}(r,\varphi) = \frac{1}{\sqrt{32\pi}} r\cos\varphi e^{-r/2}$ (que és la funció considerada en l'exemple anterior expressada en coordenades polars) per a n valors de r entre 0 i 2 (sense incloure el 0) i m valors de φ entre 0 i 2π (sense incloure el 2π, que és equivalent al 0).

3.2 Integració numèrica

Una integral definida d'una funció $f(x)$ d'una variable real x es pot resoldre numèricament subdividint l'àrea d'integració en un nombre suficientment gran de figures geomètriques senzilles (rectangles, trapezis, etc.) i sumant les seves àrees.

El mètode més senzill és el dels rectangles, que consisteix a dividir l'interval d'integració (a,b) en n subdivisions iguals de longitud $h = (b-a)/n$ i sumar les àrees dels n rectangles de base h i altura $f(x_i)$, amb $x_1 = a+h$, $x_2 = a+2h$, ..., $x_{n-1} = a+(n-1)h$, $x_n = a+nh = b$ (figura 3.1):

$$\int_a^b f(x)dx \approx hf(x_1) + hf(x_2) + \ldots + hf(x_n) = h\sum_{i=1}^{n} f(x_i) \tag{3.1}$$

L'error comès en aproximar la integral mitjançant aquesta expressió és la suma de les àrees de la part de cada rectangle que queda per sobre de la corba, i tendeix a zero quan $n \to \infty$. També es pot agafar com a altura de cada rectangle el valor de la funció en l'extrem esquerre de la base, la qual cosa condueix a l'expressió:

$$\int_a^b f(x)dx \approx hf(x_0) + hf(x_1) + \ldots + hf(x_{n-1}) = h\sum_{i=0}^{n-1} f(x_i) \tag{3.2}$$

on $x_0 = a$.

El mètode dels trapezis consisteix a sumar les àrees dels trapezis que resulten si, en cada rectangle del mètode anterior, substituïm el costat superior pel segment rectilini que uneix els punts $(x_{i-1}, f(x_{i-1}))$ i $(x_i, f(x_i))$. Aquesta suma es pot expressar de la manera següent:

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[f(x_0) + 2f(x_1) + \ldots + 2f(x_{n-1}) + f(x_n)\right] = \frac{h}{2}\left\{f(x_0) + 2\left[\sum_{i=1}^{n-1} f(x_i)\right] + f(x_n)\right\}$$

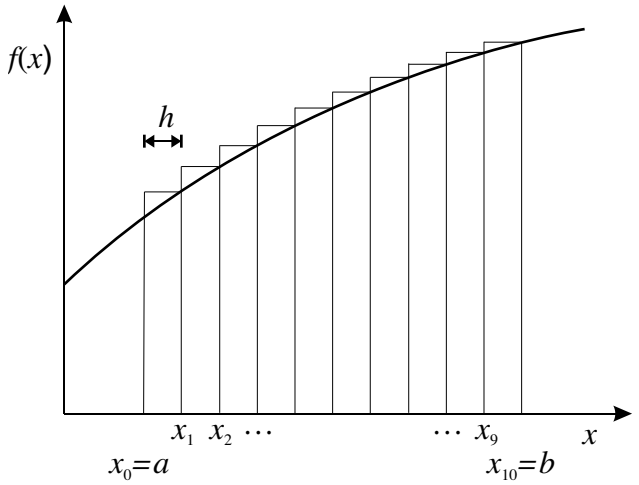

Figura 3.1: Integració numèrica pel mètode dels rectangles.

En el mètode de Simpson, el costat superior dels rectangles se substitueix per un fragment de paràbola proper al fragment corresponent de la corba $f(x)$, la qual cosa condueix a l'expressió següent:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right]$$

on el nombre n de subdivisions ha de ser parell.

Exercici 3.7
Feu un programa que calculi la integral de la funció $f(x) = \frac{2}{\sqrt{\pi}}e^{-x^2}$ entre 0 i 1 fent servir el mètode dels rectangles. El nombre de subdivisions ha de ser una dada a introduir per l'usuari, de manera que aquest pugui anar augmentant aquest nombre fins que el valor de la integral s'estabilitzi. Feu el càlcul amb les dues expressions (3.1) i (3.2) i escriviu els dos resultats juntament amb la seva mitjana. *Suggeriment*: Si s'augmenta massa, el nombre de subdivisions pot empitjorar el resultat per l'acumulació d'errors en les sumes; per prevenir-ho convé utilitzar variables de precisió doble (π = 3,1415926535897932...).
Resultat: 0,842701.

Exercici 3.8
Repetiu l'exercici anterior fent servir el mètode dels trapezis i compareu els valors obtinguts mitjançant els dos mètodes quan el nombre de subdivisions és igual a 10, a 100 i a 1000.

Exercici 3.9
Repetiu l'exercici anterior fent servir el mètode de Simpson i compareu els valors obtinguts mitjançant els dos mètodes quan el nombre de subdivisions és igual a 10, a 100 i a 1000. El programa ha de verificar que el nombre de subdivisions introduït per l'usuari sigui parell.

Exercici 3.10
El treball efectuat sobre un gas quan es comprimeix des d'un volum V_1 fins a un altre V_2 isotèrmicament i reversiblement és $w_T = -\int_{V_1}^{V_2} PdV = \int_{V_2}^{V_1} PdV$. Escriviu un programa que calculi aquest treball en joules per a un gas utilitzant l'equació d'estat de van der Waals (vegeu l'exercici 2.9 b), pàgina 32) i resolent numèricament la integral. Les dades que ha de proporcionar l'usuari són els paràmetres a i b de van der Waals, el nombre de mols, la temperatura absoluta en K, els volums V_1 i V_2 en litres i el nombre de subdivisions de l'interval d'integració. Compareu el resultat amb el que s'obté per a un gas ideal: $w_T^{id} = -\int_{V_1}^{V_2} \frac{nRT}{V} dV = nRT \ln \frac{V_1}{V_2}$. *Suggeriment*: Per convertir a joules un treball expressat en bar×l s'ha de multiplicar aquest per 100. *Comprovació* : Per a una compressió d'1,5 mols de N_2 a 273,15 K, amb els paràmetres indicats en l'exercici 2.9, des de 2 l fins a 1 l, el treball calculat amb l'equació de van der Waals és de 2310,57 J i l'obtingut amb l'equació d'un gas ideal és de 2361,31 J. Per a una expansió des d'1 l fins a 2 l amb els mateixos valors de les altres dades s'obtenen els mateixos resultats canviats de signe.

Exercici 3.11
a) Quan un àtom d'hidrogen està en el seu estat de mínima energia ($1s$), la probabilitat que en mesurar la posició de l'electró el trobem dins una esfera de radi R centrada al nucli és igual a la integral sobre aquesta esfera de la densitat de probabilitat (vegeu l'exercici 2.13, pàgina 34):

$$P_R = \int_{r=0}^{R} \int_{\theta=0}^{\pi} \int_{\varphi=0}^{2\pi} [\phi_{1s}(r)]^2 \, r^2 \sin\theta dr d\theta d\varphi = 4\pi \int_{r=0}^{R} \frac{1}{\pi a_0^3} e^{-2r/a_0} r^2 dr$$

on $r^2 \sin\theta$ és el jacobià de la transformació a coordenades esfèriques. Si fem el canvi $r/a_0 = u$,

$$P_R = 4 \int_{u=0}^{R/a_0} e^{-2u} u^2 du$$

Escriviu un programa que calculi numèricament aquesta probabilitat amb precisió doble per a un valor de R/a_0 donat per l'usuari, tot resolent la integral corresponent amb el mètode de Simpson. Calculeu les probabilitats corresponents a esferes de radis a_0, $2a_0$, $3a_0$ i $4a_0$; és a dir, per a R/a_0 = 1, 2, 3 i 4. Cap a quin valor creus que ha de tendir aquesta probabilitat quan augmenta R? *Comprovació*: Per a $R/a_0 = 1$ la probabilitat és 0,323324.
b) Si no heu utilitzat funcions externes per fer el programa anterior, modifiqueu-lo de manera que el programa principal cridi a una funció externa que calculi la integral d'una funció $f(x)$ qualsevol entre els límits a i b. Aquesta funció externa ha de cridar a una altra on es defineixi la funció $f(x)$.

3.3 Variables indexades

En aquesta secció considerarem alguns aspectes de les variables indexades que no han estat tractats fins ara.

3.3.1 Índexs amb valor mínim diferent d'1

De vegades interessa que l'índex dels elements d'un vector comenci a un valor diferent d'1, la qual cosa es pot fer declarant el vector de la manera següent:

```
dimension vect1(min1:max1), vect2(min2:max2) ...
```

on *min1* i *max1* són els valors mínim i màxim de l'índex del vector *vect1*, etc. Per exemple, si, en l'exercici 3.2 (pàgina 80) volguéssim guardar els valors de x en un vector i els de e^x en un altre (per fer qualsevol tractament posterior amb aquests valors) podríem emprar dos vectors, un per guardar els valors de x i l'altre per als de y. Com que aquests vectors han de tenir $n + 1$ components, podem fer que els seus primers components tinguin índex 0:

```
dimension x(0:1000), y(0:1000)
write (6,*) "Entra el nombre de subdivisions (maxim 1000)"
read (5,*) n
h = 2./n
do i=0,n
  x(i) = i*h
  y(i) = exp(x(i))
  write (6,*) x(i), y(i)
end do
...
```

El mateix s'aplica quan es declaren vectors mitjançant instruccions de declaració de tipus de variables (`integer`, `real`, etc.).

Exercici 3.12
Indiqueu la quantitat de memòria RAM, expressada en bytes i en kiB, que ocuparan durant l'execució les variables declarades mitjançant les instruccions següents:

```
integer a(50), b(-10:10)
real nombre, dades(-40:0)
character*10 llista(80)
double precision x(0:1000), y(0:1000)
```

Solució: 17268 bytes $\approx$ 16,863 kiB.

Exercici 3.13
Feu un programa que generi una taula amb els valors d'abscisses i d'ordenades de la corba $\sin(x)$ per a $x \in [-\pi, \pi]$, tot dividint aquest interval en 200 parts iguals (podeu aprofitar l'exercici 3.1). Els valors de x i de $\sin(x)$ s'han de guardar en sengles vectors, i el programa ha de cridar a una funció externa que en calculi el producte escalar (vegeu l'exercici 2.105, pàgina 77). Feu servir precisió doble (π = 3,1415926535897932...). *Resultat*: 199,984.

Exercici 3.14
Feu un programa que busqui una arrel real d'un polinomi de grau $n \leq 50$ en x. Podeu aplicar l'algorisme de Newton-Raphson (vegeu l'exercici 2.56, pàgina 52) a l'equació:

$$a_0 + a_1 x + a_2 x^2 + \ \ldots \ + \ a_n x^n = 0$$

L'usuari haurà d'introduir el grau de l'equació (n), els coeficients a_0, a_1, a_2, ..., a_n, una primera estimació de la solució buscada (x_0) i un valor petit ε que determini la convergència del procés iteratiu. Utilitzeu dues funcions externes, una per avaluar el polinomi i l'altra per avaluar la seva derivada. Feu que el programa s'aturi si el nombre d'iteracions és superior a un cert valor *itmax*, subministrat també per l'usuari, o si el valor absolut de la derivada és inferior a ε. *Comprovació*: Feu servir les dades indicades en l'exercici 2.56.

Els índexs de les matrius també poden començar amb valors diferents d'1, la qual cosa s'haurà d'indicar en la declaració:

```
dimension m1(mif1:maf1,mic1:mac1), m2(mif2:maf2,mic2:mac2) ...
```

on *mif1*, *maf1* i *mic1*, *mac1* són els valors mínim i màxim dels índexs de files i de columnes de la matriu *m1*, etc. En lloc de la instrucció `dimension` pot haver-hi qualsevol declaració (explícita) de tipus de dades (`integer`, `real`, etc.). Per exemple, la instrucció:

```
real f(0:100,0:100), g(-10:10,0:30), h(10,30)
```

declara una matriu `f` de 101 files i 101 columnes, una altra matriu `g` de 21 files i 31 columnes i una tercera `h` de 10 files i 30 columnes. La matriu `g`, per exemple, tindrà la forma següent:

$$\begin{pmatrix} g_{-10,0} & g_{-10,1} & \cdots & g_{-10,30} \\ \cdots & \cdots & \cdots & \cdots \\ g_{10,0} & g_{10,1} & \cdots & g_{10,30} \end{pmatrix}$$

Exercici 3.15
Indiqueu la quantitat de memòria RAM, expressada en bytes i en kiB, que ocuparan durant l'execució les variables declarades mitjançant la instrucció de declaració de l'exemple anterior.
Solució: 44608 bytes = 43,5625 kiB.

3.3.2 Component més gran o més petit d'un vector

Per trobar el component més gran d'un vector podem emprar el mateix algorisme que utilitzem quan ho fem a mà. Per exemple, si tenim una pila d'exàmens i volem saber quina és la nota més alta mirarem la del primer examen i la retindrem en la nostra memòria; després mirarem la del segon examen i, si és més alta que la del primer, la retindrem al cap en el lloc de la primera. Així, anirem mirant, una per una, totes les notes i, cada vegada que en trobem una de superior a la que tenim al cap —que serà la més alta de les que portem mirades—, la memoritzarem en substitució d'aquella. En acabar-se la pila, la nota retinguda en la nostra memòria serà la més alta.

Exercici 3.16
a) Feu un programa que llegeixi el nombre d'alumnes presentats a un examen i les seves notes (vegeu l'exemple de la pàgina 61), i busqui i escrigui la nota més alta. Comproveu que funciona tant en el cas que la primera nota no sigui la més alta com en el cas que ho sigui.
b) Modifiqueu el programa anterior perquè llegeixi també els cognoms dels alumnes i indiqui qui ha tret la nota més alta.
c) Modifiqueu el programa anterior perquè, en el cas que hi hagi més d'un alumne amb la nota més alta, els escrigui tots.
d) Modifiqueu el programa anterior perquè indiqui també quina és la nota més baixa.

3.3.3 Ordenació dels components d'un vector

Suposem que tenim un programa per gestionar les notes d'un examen (confeccionar llistes, fer estadístiques, etc.) i volem que tregui una llista de notes ordenades de la més alta a la més baixa. Hi ha molts algorismes per fer ordenacions d'aquest tipus. Alguns d'aquests generen, a partir d'un vector, un de nou amb les notes ordenades; en canvi, d'altres fan la reordenació intercanviant valors entre els components del mateix vector original. Entre els algorismes del segon tipus, un dels més senzills consisteix a buscar la nota més alta i, si no és la primera de la llista, intercanviar-la amb aquesta; després es busca la nota més alta de les restants (de la segona a l'última) —o sigui, la segona més alta de totes— i, si és major que la segona de la llista, es bescanvia amb aquesta. Aquest procés es continua fins a col·locar la penúltima nota més alta (la segona més baixa) en la penúltima posició del vector. La nota més baixa haurà quedat en l'última posició.

A continuació presentem un esquema en pseudocodi del programa d'ordenació:

Programa Ordenació
  (declaració del vector *nota*)
  llegir el nombre d'alumnes (*numal*)
  llegir els *numal* elements de *nota*
  ordenar els *numal* elements de *nota*
  escriure els *numal* elements de *nota*
fi programa

Com que els algorismes per llegir i escriure vectors són prou coneguts (vegeu l'apartat 2.11.1), no caldrà detallar-los més abans de fer la transcripció del pseudocodi al Fortran. L'ordenació dels elements de *nota* és una acció més complexa que sí que caldrà detallar:

```
acció ordenar els numal elements de nota
  per a i des de 1 fins a numal − 1 fer
  |  buscar la "nota més alta" de nota(i + 1) fins a nota(numal)
  |  si "nota més alta" > nota(i) llavors
  |  |  intercanviar "nota més alta" amb nota(i)
  |  fi si
  fi per a
fi acció
```

El símil dels gots utilitzat en l'exercici 2.4 (pàgina 31) per il·lustrar un algorisme d'intercanvi dels valors de dues variables s'estén fàcilment a aquest cas i pot ajudar a clarificar la diferència entre l'índex d'un component i el valor d'aquest. Considerem una filera de, per exemple, 10 gots, numerats de l'1 fins al 10, amb diferents volums d'aigua. La filera representaria un vector i cada got seria un dels seus components. El volum d'aigua contingut en un got seria el valor emmagatzemat en el component, i l'índex d'aquest seria el número del got:

número del got:	1	2	3	...	10
volum d'aigua en cm^3 :	25	12	34	...	29

Per reordenar els continguts dels gots de manera que el més ple sigui el que té el número 1 i el menys ple el que té el número 10, podem començar comparant el volum del primer amb el dels altres. Si el got més ple no és el primer, intercanviarem el contingut d'aquest amb el d'aquell. Així, si el més ple és el tercer, intercanviarem el contingut d'aquest amb el del primer got, per a la qual cosa haurem d'utilitzar un got auxiliar buit. Llavors la filera de gots quedarà de la manera següent:

número del got:	1	2	3	...	10
volum d'aigua en cm^3 :	34	12	25	...	29

A continuació repetirem el procés per trobar el segon got més ple i intercanviar el seu contingut amb el del got número 2, i així successivament fins a col·locar el penúltim més ple en el got número 9, de manera que el got número 10 sigui el més buit.

Exercici 3.17
Feu un programa que llegeixi el nombre d'alumnes presentats a un examen i el cognom i la nota de cadascun d'ells i tregui una llista de notes ordenades de major a menor, amb el cognom al davant de cadascuna. Per determinar l'índex del component més gran dels que segueixen a l'i-èsim feu servir una funció externa de dues variables: el vector de notes i el valor de i (vegeu l'exercici 3.16). Tingueu en compte que cada vegada que s'intercanvien un parell de notes s'han d'intercanviar també els cognoms corresponents.

Un algorisme més breu que el descrit abans però que, en general, implica un nombre més gran d'intercanvis és el següent:

```
acció ordenar els numal elements de nota
  per a i des de 1 fins a numal − 1 fer
  |  per a j des de i + 1 fins a numal fer
  |  |  si nota(j) > nota(i) llavors
  |  |  |  intercanviar nota(j) amb nota(i)
  |  |  fi si
  |  fi per a
  fi per a
fi acció
```

Exercici 3.18
a) Feu un programa que llegeixi el nombre de components n d'un vector V i els seus valors i cridi a una subrutina que els ordeni en ordre decreixent emprant l'algorisme anterior. En acabar aquesta, el programa principal ha d'escriure el vector ordenat.
b) Modifiqueu el programa de l'apartat anterior per tal que els intercanvis entre valors de components dels vectors es facin mitjançant una crida a una segona subrutina que bescanviï els valors de dues variables passades com a arguments; és a dir, la subrutina d'ordenació de components ha de cridar a una subrutina d'intercanvi de variables. *Observació* : En aquesta subrutina, els valors a intercanviar han de ser variables escalars (no vectors); per això els corresponents paràmetres de la instrucció de crida han de ser components concrets del vector (cada un dels quals és un escalar), no el vector sencer.

3.3.4 Producte de matrius

El producte d'una matriu A, de m files i l columnes, per una altra B, de l files i n columnes, és una nova matriu C, de m files i n columnes:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1l} \\ a_{21} & a_{22} & \dots & a_{2l} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{ml} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{l1} & b_{l2} & \dots & b_{ln} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{pmatrix}$$

els elements de la qual es defineixen de la manera següent:

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} + \ldots + a_{1l}b_{l1} = \sum_{k=1}^{l} a_{1k}b_{k1} \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22} + \ldots + a_{1l}b_{l2} = \sum_{k=1}^{l} a_{1k}b_{k2} \\ &\dots \\ c_{mn} &= a_{m1}b_{1n} + a_{m2}b_{2n} + \ldots + a_{ml}b_{ln} = \sum_{k=1}^{l} a_{mk}b_{kn} \end{aligned}$$

és a dir,

$$c_{ij} = \sum_{k=1}^{l} a_{ik}b_{kj} \qquad \text{on} \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n \tag{3.3}$$

A continuació presentem un primer esquema en pseudocodi de l'algorisme per efectuar aquesta operació:

Programa Producte de matrius
(declaració de les matrius A, B i C)
llegir els nombres de files i columnes de A i de B (m, l, i n)
llegir els elements de A
llegir els elements de B
calcular la matriu C, producte de A per B
escriure la matriu C
fi programa

La part més complicada d'aquest algorisme és el càlcul de la matriu C. Aquest s'ha de fer element a element (equació (3.3)), la qual cosa suggereix la utilització de dues estructures repetitives de tipus *per a* aniuades. Per exemple, podem fer variar l'índex de les files de C (i) en l'estructura externa i el de les columnes (j) en la interna; dit de manera abreujada, calcularem els elements de C *per files*:

```
acció calcular matriu C
  per a i des de 1 fins a m fer
  |  per a j des de 1 fins a n fer
  |  |  calcular c_ij
  |  fi per a
  fi per a
fi acció
```

com que l'acció "calcular c_{ij}" encara és prou complicada, convindrà desenvolupar-la una mica més. D'acord amb l'equació (3.3), c_{ij} es calcula com un sumatori que ha d'anar acumulant els productes $a_{ik}b_{kj}$ per als successius valors de k (vegeu l'apartat 2.10.2):

```
acció calcular c_ij
  c_ij ← 0
  per a k des de 1 fins a l fer
  |  c_ij ← c_ij + a_ik b_kj
  fi per a
fi acció
```

Exercici 3.19
Feu un programa que llegeixi una matriu real A de m files i l columnes, i una altra B de l files i n columnes, essent m, l, i n dades a introduir per l'usuari (amb valors màxims de 100), calculi la matriu C producte de A per B i l'escrigui per files. *Comprovació*: $\begin{pmatrix} 1 & 2 & 3 \\ -2 & -3 & -4 \end{pmatrix} \times \begin{pmatrix} 1 & 2 & 3 & 4 \\ -2 & -3 & -4 & -5 \\ 3 & 4 & 5 & 6 \end{pmatrix} = \begin{pmatrix} 6 & 8 & 10 & 12 \\ -8 & -11 & -14 & -17 \end{pmatrix}$.

Exercici 3.20
Modifiqueu el programa anterior perquè faci servir les subrutines de lectura i escriptura de matrius de l'exercici 2.99 (pàgina 74) i una tercera subrutina que multipliqui dues matrius.

Exercici 3.21
Feu un programa que demani la dimensió m i els elements d'un vector columna real B, i els elements d'una matriu real quadrada A de $m \times m$. Després ha de multiplicar la matriu pel vector i escriure el vector columna resultant. *Comprovació*: $\begin{pmatrix} 1 & 2 & 3 \\ -2 & -3 & -4 \\ 3 & 4 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 8 \\ -12 \\ 16 \end{pmatrix}$

3.3.5 Diagonalització de matrius

La diagonalització de matrius o, el que és equivalent, el càlcul de vectors i valors propis de matrius és un problema molt important en la física, la química i la tecnologia. En molts casos les matrius que s'han de diagonalitzar són reals i simètriques, i s'han desenvolupat mètodes molt eficients per diagonalitzar-les. Una matriu F quadrada, real i simètrica ($f_{ji} = f_{ij}$ per a tot i i tot j) es pot transformar en una matriu diagonal G fent l'operació $\mathsf{C}^t\ \mathsf{F}\ \mathsf{C} = \mathsf{G}$, on C és la matriu de transformació (les columnes de la qual són els vectors propis de F) i C^t és la matriu transposada de C (vegeu l'exercici 2.71, pàgina 60):

$$\begin{pmatrix} c_{11} & c_{21} & \dots & c_{n1} \\ c_{12} & c_{22} & \dots & c_{n2} \\ \dots & \dots & \dots & \dots \\ c_{1n} & c_{2n} & \dots & c_{nn} \end{pmatrix} \times \begin{pmatrix} f_{11} & f_{12} & \dots & f_{1m} \\ f_{21} & f_{22} & \dots & f_{2m} \\ \dots & \dots & \dots & \dots \\ f_{n1} & f_{n2} & \dots & f_{nm} \end{pmatrix} \times \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix} = \begin{pmatrix} g_1 & 0 & \dots & 0 \\ 0 & g_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & g_n \end{pmatrix}$$

Cada element diagonal g_i és el valor propi de F corresponent al vector propi de components $c_{1i}, \ldots, c_{ni}$. Es pot demostrar que C és una matriu ortogonal, és a dir:

$$\mathsf{C}^{-1} = \mathsf{C}^t$$

on C^{-1} és la matriu inversa de C ($\mathsf{C}^{-1}\mathsf{C} = \mathsf{C}\mathsf{C}^{-1} = \mathsf{1}$).

En aquest apartat considerarem un mètode molt senzill per trobar la transformació que diagonalitza una matriu quadrada, real i simètrica: el *mètode de Jacobi*. Si la matriu F que s'ha de diagonalitzar és de dimensions 2×2 és fàcil comprovar que es pot diagonalitzar mitjançant la matriu C de transformació següent:

$$\begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \qquad \text{amb} \qquad \tan 2\alpha = \frac{2f_{12}}{f_{11} - f_{22}}$$

Exercici 3.22
a) Feu un programa que llegeixi una matriu 2×2 real i simètrica, comprovi que efectivament és simètrica, la diagonalitzi i escrigui la matriu diagonalitzada, els valors propis i la matriu de transformació. *Comprovació*: Els valors propis de la matriu $\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$ són $-0,236068$ i $4,23607$, i els vectors propis corresponents són $\begin{pmatrix} 0,850651 \\ -0,525731 \end{pmatrix}$ i $\begin{pmatrix} 0,525731 \\ 0,850651 \end{pmatrix}$ (els dos components de qualsevol d'aquests vectors poden tenir el signe canviat). *Suggeriment*: Empreu una subrutina basada en el programa proposat en l'exercici 3.19 per calcular els productes de matrius.

Si la matriu és de dimensions majors que 2 es pot utilitzar el procediment iteratiu següent:

- es busca l'element f_{ij} no diagonal del triangle superior ($i < j$) que tingui el valor absolut més gran;
- s'efectua la transformació $\mathsf{C}_1^t\ \mathsf{F}\ \mathsf{C}_1 = \mathsf{F}_1$ amb una matriu de transformació que només difereix de la matriu identitat (uns a la diagonal principal i zeros a la resta) en les files i columnes i-èsimes i j-èsimes:

$$\mathsf{C}_1 = \begin{pmatrix} 1 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & 0 & \cos\alpha & 0 & \ldots & 0 & -\sin\alpha & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & 0 & \sin\alpha & 0 & \ldots & 0 & \cos\alpha & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 1 \end{pmatrix} \begin{matrix} \ldots \\ \ldots \\ \longleftarrow \text{fila } i \\ \ldots \\ \longleftarrow \text{fila } j \\ \ldots \\ \ldots \end{matrix}$$

$$\begin{matrix} & & & \uparrow & & & & \uparrow & & & \\ \ldots & \ldots & \ldots & \text{colu.}i & \ldots & \ldots & \ldots & \text{colu.}j & \ldots & \ldots & \ldots \end{matrix}$$

  on $\tan 2\alpha = 2f_{ij}/(f_{ii} - f_{jj})$; la nova matriu F_1 té els elements $(\mathsf{F}_1)_{ij}$ i $(\mathsf{F}_1)_{ji}$ iguals a zero (de fet, en efectuar la transformació es comprova que aquesta només modifica les files i columnes i-èsimes i j -èsimes de F);
- es repeteix el procés amb la matriu F_1.

A cada pas del procediment iteratiu es fan zero dos elements no diagonals de la matriu F_i però, en modificar-se les files i columnes corresponents, poden deixar d'anul·lar-se elements d'aquestes que s'havien anul·lat en iteracions anteriors. No obstant això, l'elecció en cada iteració de l'element no diagonal de valor absolut més gran garanteix, normalment, que tots els elements no diagonals tendeixin cap a zero a mesura que avança el procés iteratiu. Acabarem el procés quan no hi hagi cap element no diagonal de F_i que tingui un valor absolut superior a una constant ε prou petita (per exemple, 10^{-6} si utilitzem variables de precisió senzilla). Si han calgut p iteracions la matriu de transformació buscada serà:

$$\mathsf{C} = \mathsf{C}_1 \times \mathsf{C}_2 \times \ldots \times \mathsf{C}_p$$

ja que

$$\begin{aligned}\mathsf{F}_p &= \mathsf{C}_p^t\mathsf{F}_{p-1}\mathsf{C}_p = \mathsf{C}_p^t\left(\mathsf{C}_{p-1}^t\mathsf{F}_{p-2}\mathsf{C}_{p-1}\right)\mathsf{C}_p = \dots \\ &= \mathsf{C}_p^t\left(\mathsf{C}_{p-1}^t\left(\dots\left(\mathsf{C}_1^t\mathsf{F}\mathsf{C}_1\right)\dots\right)\mathsf{C}_{p-1}\right)\mathsf{C}_p = \left(\mathsf{C}_1\dots\mathsf{C}_{p-1}\mathsf{C}_p\right)^t\mathsf{F}\left(\mathsf{C}_1\dots\mathsf{C}_{p-1}\mathsf{C}_p\right)\end{aligned}$$

En un primer nivell d'estructuració l'algorisme per resoldre aquest problema es podria expressar de la manera següent:

Programa Diagonalització d'una matriu simètrica
(declaració de les matrius F i C i dues matrius auxiliars A i CT)
llegir el nombre de files (o columnes) de F
llegir els elements de F
comprovar que F és simètrica
diagonalitzar F, tot calculant C
escriure la matriu F diagonalitzada, els valors propis i la matriu C
fi programa

La més complicada d'aquestes accions és, sens dubte, la diagonalització de F, que implicarà un procés iteratiu amb nombre d'iteracions indeterminat:

acció diagonalitzar F, tot calculant C
llegir el valor de ε
inicialitzar C a la matriu identitat
buscar els índexs *imax* i *jmax* de l'element no diagonal del triangle
superior de F més gran en valor absolut
mentre $|f_{imax,jmax}| > \varepsilon$ fer
| construir la matriu de transformació parcial C_k
| construir una matriu CT que sigui igual a C_k^t
| calcular la matriu $\mathsf{A} = \mathsf{CT}\times\mathsf{F}$
| calcular la matriu $\mathsf{F} = \mathsf{A}\times\mathsf{C}_k$
| calcular la matriu $\mathsf{A} = \mathsf{C}\times\mathsf{C}_k$
| copiar la matriu A en la matriu C
| buscar els índexs *imax* i *jmax* de l'element no diagonal del triangle
| superior de F més gran en valor absolut
fi mentre
fi acció

Deixem que el lector acabi de detallar aquest esquema per resoldre l'exercici següent.

Exercici 3.23
a) Feu un programa que llegeixi una matriu $n \times n$ real i simètrica, essent n una dada a introduir per l'usuari, comprovi que és simètrica, la diagonalitzi (s'haurà de llegir també el valor de ε) i escrigui la matriu diagonalitzada, els seus valors propis i la matriu de transformació. *Suggeriment*: Feu servir subrutines per llegir, multiplicar i escriure matrius. *Comprovació*: Els valors propis de la matriu $\begin{pmatrix} 1 & 2 & -3 \\ 2 & 3 & 4 \\ -3 & 4 & 5 \end{pmatrix}$ són $-3,07851$, 3,73953 i $8,33899$ i els vectors propis corresponents són[a] $\begin{pmatrix} -0,652129 \\ 0,554654 \\ -0,516804 \end{pmatrix}$, $\begin{pmatrix} 0,734807 \\ 0,630167 \\ -0,250897 \end{pmatrix}$ i $\begin{pmatrix} -0,186512 \\ 0,543368 \\ 0,818514 \end{pmatrix}$.
b) Modifiqueu el programa de l'apartat anterior de manera que el programa principal cridi a una subrutina per diagonalitzar la matriu. Aquesta ha de rebre la matriu a diagonalitzar i tornar-la diagonalitzada, juntament amb la matriu de transformació.

[a] El signe dels vectors propis no està determinat, de manera que si algun dels vectors que obteniu té els signes dels tres components canviats, la solució és correcta. D'altra banda, si hi ha valors propis degenerats (iguals), els vectors propis corresponents no estan unívocament determinats.

c) Modifiqueu el programa de l'apartat anterior perquè la lectura de la matriu a diagonalitzar es faci mitjançant una subrutina que llegeixi només els elements del triangle superior i assigni els elements restants (vegeu l'exercici 2.80, pàgina 63). D'aquesta manera es podrà ometre la comprovació que la matriu és simètrica.

Exercici 3.24
Donada una matriu quadrada, real i simètrica F de $n \times n$, la seva matriu inversa F^{-1} es por calcular, si existeix, diagonalitzant la matriu F ($\mathsf{C}^t\ \mathsf{F}\ \mathsf{C} = \mathsf{G}$), construint una altra matriu diagonal els elements no nuls de la qual siguin els inversos dels valors propis de F ($1/g_{11},\ \ldots,\ 1/g_{nn}$), i aplicant a la matriu resultant (G^{-1}) la transformació inversa a la que diagonalitza F ($\mathsf{F}^{-1} = \mathsf{C}\mathsf{G}^{-1}\mathsf{C}^t$).
a) Feu un programa que demani la dimensió i els elements d'una matriu quadrada, real i simètrica F i, si no hi ha cap valor propi nul, calculi i escrigui F^{-1}. *Suggeriment*: Aprofiteu tot el que pugueu de l'exercici anterior. *Comprovació*: $\begin{pmatrix} 7 & -1 & -1 \\ -1 & 5 & 1 \\ -1 & 1 & 5 \end{pmatrix}^{-1} = \begin{pmatrix} 0,1500 & 0,0250 & 0,0250 \\ 0,0250 & 0,2125 & -0,0375 \\ 0,0250 & -0,0375 & 0,2125 \end{pmatrix}$.
b) Modifiqueu el programa de l'apartat anterior perquè calculi els productes $\mathsf{F}\mathsf{F}^{-1}$ i $\mathsf{F}^{-1}\mathsf{F}$ i els escrigui, de manera que l'usuari pugui comprovar que aquests productes són iguals a la matriu identitat.

Exercici 3.25
En certes àrees de la ciència, com ara la mecànica quàntica, s'han de calcular matrius que són funcions d'altres matrius. Per exemple, l'exponencial e^{F} d'una matriu quadrada, real i simètrica F és una nova matriu, de les mateixes dimensions que F, que es pot obtenir de la manera següent: es diagonalitza la matriu F ($\mathsf{C}^t\ \mathsf{F}\ \mathsf{C} = \mathsf{G}$), es construeix una altra matriu diagonal (e^{G}) els elements no nuls de la qual siguin les exponencials dels valors propis de F ($e^{g_{11}},\ \ldots,\ e^{g_{nn}}$), i s'aplica a la matriu resultant la transformació inversa a la que diagonalitza F ($e^{\mathsf{F}} = \mathsf{C}e^{\mathsf{G}}\mathsf{C}^t$). Feu un programa que demani la dimensió i els elements d'una matriu quadrada, real i simètrica F i calculi i escrigui la matriu e^{F}. *Comprovació*: $e^{\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}} = \begin{pmatrix} 19,6800 & 30,5651 \\ 30,5651 & 50,2452 \end{pmatrix}$.

Una aplicació de la diagonalització de matrius d'interès en química és el càlcul d'orbitals moleculars i, en particular, el d'orbitals π de molècules conjugades mitjançant el mètode aproximat de Hückel. Aquestes són molècules planes amb un conjunt d'àtoms (normalment C, N, O ...) units per enllaços senzills i dobles alternats —el sistema conjugat— i àtoms d'hidrogen. Els seus orbitals moleculars es poden classificar com a σ (simètrics respecte del pla de la molècula) o π (antisimètrics respecte del pla de la molècula). D'acord amb el mètode de Hückel, els orbitals π s'obtenen com a vectors propis d'una *matriu de Hückel* i els valors propis corresponents són les energies d'aquells orbitals.

Considerarem només el cas dels hidrocarburs conjugats, molècules formades per àtoms de carboni i d'hidrogen. Cada fila o columna de la matriu de Hückel (H) correspon a un àtom del sistema π, el qual, en aquest cas, està format només per carbonis. Els elements diagonals (H_{rr}) són nuls (si s'escull un origen d'energies adient) i els no diagonals (H_{rs}) també ho són si els seus índexs corresponen a àtoms de carboni (C_r i C_s) no enllaçats directament. Si els índexs corresponen a carbonis enllaçats —amb enllaç senzill o doble—, l'element val 1 en les unitats d'energia adequades. Per exemple, la matriu de Hückel del benzè, amb els sis carbonis de l'anell numerats correlativament de l'1 al 6 (figura 3.2), és:

$$\mathsf{H} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

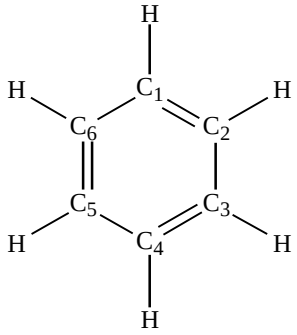

Figura 3.2: Numeració dels àtoms del sistema π del benzè.

L'equació de valors propis d'aquesta matriu és:

$$\mathsf{H}\mathsf{c}_i = \epsilon_i \mathsf{c}_i \qquad i = 1, \ldots, 6 \tag{3.4}$$

Cada vector propi (c_i) determina un orbital molecular π (ϕ_i), el qual s'expressa com una combinació lineal d'orbitals atòmics de tipus $2p_z$ —que anomenarem χ_r— centrats a cada carboni:

$$\phi_i = \sum_{r=1}^{6} c_{ri}\chi_r \qquad i = 1, \ldots, 6$$

El valor propi corresponent (ϵ_i) és l'energia de l'orbital molecular. Les sis equacions (3.4) es poden agrupar en una de sola introduint una matriu quadrada C les columnes de la qual siguin c_1, ..., c_6 i una matriu diagonal E els elements diagonals de la qual siguin ϵ_1, ..., ϵ_6 :

$$\mathsf{HC} = \mathsf{CE}$$

Multiplicant per $\mathsf{C}^{-1} = \mathsf{C}^t$ per l'esquerra es veu que aquesta equació equival a la de diagonalització de la matriu de Hückel:

$$\mathsf{C}^t\mathsf{HC} = \mathsf{E}$$

Exercici 3.26
Feu un programa que calculi els coeficients dels orbitals moleculars π d'un hidrocarbur conjugat i les energies orbitals corresponents emprant el mètode de Hückel. *Suggeriment*: El programa haurà de demanar a l'usuari el nombre d'àtoms de carboni que formen el sistema conjugat. Per construir la matriu de Hückel podeu assignar el valor 0 a tots els seus elements i, després, demanar a l'usuari que entri els índexs $\{r, s\}$ de cada parell de carbonis enllaçats, per tal d'assignar un 1 als elements H_{rs} i H_{sr} corresponents. *Suggeriment*: Aprofiteu tot el que pugueu de l'exercici 3.23 . *Comprovació*: Les energies dels orbitals del benzè són -2, -1, -1, 1, 1 i 2.

3.3.6 Variables indexades en subprogrames

En la secció 2.13 hem dit que les variables indexades que es passen com a arguments a un subprograma s'han de declarar amb les mateixes dimensions en el subprograma i en el (sub)programa que el crida. Fins ara, aquestes dimensions s'han especificat sempre mitjançant l'indicació explicíta del seu valor, la qual cosa té un inconvenient: si decidim canviar les dimensions declarades per a alguna d'aquelles variables —per exemple, augmentar el nombre de columnes d'una matriu— ho hem de fer en tots els (sub)programes on aparegui, la qual cosa pot ser laboriosa quan n'hi ha molts. Per evitar-ho es poden passar les dimensions com a paràmetres de la instrucció de crida, de manera que un canvi en les dimensions del programa que fa les crides afecti tots els subprogrames cridats. Així, per duplicar les dimensions de les variables indexades del programa:

```
parameter (maxfil=100, maxcol=80)
dimension v(maxfil), mat(maxfil,maxcol)
...
call sub1(v, mat, maxfil, maxcol, nfil, ncol)
...
z = fun1(mat, maxfil, maxcol, nfil, ncol)
...
end
subroutine sub1(x, m, maxf, maxc, nf, nc)
dimension x(maxf), m(maxf,maxc)
...
end
function fun1(a, mf, mc, nf, nc)
integer a(mf,mc)
...
end
```

—on `nfil` i `ncol` són els nombres de files i columnes que s'utilitzaran— només caldrà modificar la primera instrucció:

```
parameter (maxfil=200, maxcol=160)
```

De fet, encara es pot reduir més la informació especificada sobre les dimensions de les variables indexades en els subprogrames en els quals apareixen com a arguments. Si es tracta d'un vector —per exemple, `v` — no cal especificar la dimensió; n'hi ha prou amb indicar el caràcter vectorial de la variable amb qualsevol de les notacions `v(*)` o `v(1)`. Si es tracta d'una matriu, l'únic que és indispensable és fixar el nombre de files, i podem substituir el nombre de columnes per un '*' o un '1' (`m(maxf,*)`, `a(mf,1)`, etc.). Això es deu al fet que la matriu es guarda en la memòria com a vector, amb cada columna a continuació de l'anterior, i el compilador ha de saber on acaba cada una i on comença la següent.[2] Així, l'exemple anterior es podria programar de la manera equivalent:

```
parameter (maxfil=100, maxcol=80)
dimension v(maxfil), mat(maxfil,maxcol)
...
call sub1(v, mat, maxfil, nfil, ncol)
...
z = fun1(mat, maxfil, nfil, ncol)
...
end
subroutine sub1(x, m, maxf, nf, nc)
dimension x(*), m(maxf,*)
...
end
function fun1(a, mf, nf, nc)
integer a(mf,*)
...
end
```

3.4 Escriptura i lectura amb format

En la secció 2.4 hem vist que, quan un programa escriu dades en format lliure (`write (6,*) ...`), la manera en què aquestes es presenten (espais reservats per als nombres enters, nombre de xifres decimals de les dades reals, etc.) està prefixada pel compilador. Si volem presentar les dades d'una altra manera, és a dir, amb un altre format, haurem de substituir l'asterisc de la instrucció d'escriptura per una etiqueta:

[2] De fet, es podria dimensionar la variable indexada de manera diferent en el programa cridat que en el que fa la crida, encara que això no és una pràctica recomanable.

```
write (6,etiq) varoco1, varoco2, ..., varocon
```

on *etiq* és l'etiqueta d'una instrucció `format` (vegeu l'apartat 2.3.3) que ha de recollir la llista d'especificacions necessàries per indicar com s'han d'escriure les dades que apareixen en la instrucció `write`:

```
etiqueta format llista d'especificacions
```

Les especificacions d'una instrucció `format` utilitzen certs codis, alguns dels quals seran detallats en l'apartat següent, i poden anar acompanyades de cadenes de caràcters entre cometes. Els diferents ítems de la llista d'especificacions se separen per comes. Per exemple, el codi 'i' seguit d'un nombre enter 'n' indica que una dada entera s'ha d'escriure ocupant n posicions. Així, les instruccions

```
      n = 34
      m = 125
      write (6,85) n,m
   85 format (i4,i4)
```

faran que s'escrigui en el monitor (unitat 6) el text següent

```
  34 125
```

on el nombre '34' està precedit per dos espais en blanc i el nombre '125' està precedit per un espai (tots dos ocupen quatre posicions). La inclusió de cadenes de caràcters pot millorar la claredat del missatge presentat; per exemple, si substituïm la instrucció `format` en l'exemple anterior per

```
   85 format ("Primer resultat =",i4,"; segon resultat =",i4)
```

obtindrem el missatge següent

```
Primer resultat =  34; segon resultat = 125
```

Si haguéssim assignat el valor 12500 en comptes de 125 a la variable `m`, s'escriuria

```
Primer resultat =  34; segon resultat =****
```

on els asteriscs indiquen que el valor que es vol escriure no cap en l'espai que li ha reservat la instrucció `format`. En canvi, l'escriptura en format lliure utilitza les posicions que calgui per escriure cada dada; així, les instruccions

```
      write (6,*) "Primer resultat =",n,"; segon resultat =",m
```

escriuran

```
Primer resultat = 34; segon resultat = 12500
```

Es poden incloure factors de repetició davant d'una o més especificacions per no haver d'escriure-les diverses vegades; per exemple, les instruccions

```
      n = 34
      m = 125
      write (6,40) n,m
   40 format ("Resultats:",2i4)
      write (6,41) n,m
   41 format (2(" Resultat:",i4))
```

faran que s'escrigui

```
Resultats:  34 125
 Resultat:  34 Resultat: 125
```

La instrucció `format` és no executable (vegeu l'apartat 2.3.2), i pot anar en qualsevol punt del programa. Les pràctiques més habituals consisteixen a posar cada instrucció `format` a continuació de la instrucció `write` que la fa servir o ajuntar-les totes, bé al final del programa (just abans de la instrucció `end`), bé abans de la primera instrucció executable.

En un programa pot haver-hi més d'una instrucció `write` (o `read`) que faci referència a la mateixa instrucció `format`; per exemple, les instruccions

```
      n = 34
      m = 125
      write (6,32) n
      write (6,32) m
   32 format("Resultat =",i4)
```

faran que s'escrigui

```
Resultat =  34
Resultat = 125
```

La instrucció `format` pot contenir especificacions per a més o menys variables que les que s'escriuen en una instrucció `write`. En el primer cas no es tenen en compte les especificacions sobrants; així, les instruccions

```
      n = 34
      m = 125
      write (6,800) n
  800 format ("Resultats:",i4,i5)
```

faran que s'escrigui

```
Resultats:  34
```

Si el nombre d'especificacions és més petit que el de variables a escriure, quan s'esgoten les especificacions es torna a començar per la primera, de manera que les instruccions

```
      n = 34
      m = 125
      write (6,1) n,m
   1  format(" Resultat =",i4)
```

faran que s'escrigui

```
 Resultat =  34
 Resultat = 125
```

Tot i que la utilització de la instrucció `format` en relació amb la lectura de dades és menys freqüent, hi ha casos en què pot ser interessant; per exemple, per llegir dos nombres enters de 5 xifres cadascun que estan enganxats (és a dir, sense coma ni espais en blanc entre ells) en un fitxer de dades, podem fer servir les instruccions:

```
      read (10,50) n,m
  50  format(2i5)
```

Hi ha una manera alternativa d'especificar el format d'entrada o sortida sense utilitzar instruccions `format`: es tracta d'indicar el format en la mateixa instrucció `read` o `write`, al lloc on es posaria l'etiqueta. Per exemple, les instruccions

```
      n = 34
      write (6,'(" Resultat =",i4)') n
```

faran que s'escrigui en el monitor el següent

```
 Resultat =  34
```

No obstant això, la utilització d'una instrucció `format` té l'avantatge que, com ja hem vist, s'hi pot fer referència des de diverses instruccions `read` o `write`.

3.4.1 Codis per a especificacions de format

A continuació descriurem les especificacions de format que més s'utilitzen en les instruccions `format`. En tots els casos, `n` representa una constant entera positiva que indica la longitud total de l'espai on s'escriurà una dada; és a dir, el nombre d'espais reservats per a aquesta. Si el que s'ha d'escriure no cap en els `n` espais, s'escriuran `n` asteriscs (`***...`).

- `in`: s'utilitza per escriure dades enteres. Si el nombre de dígits d'una d'aquestes dades és <`n` es deixen espais en blanc a l'esquerra; és a dir, el camp imprès es justifica per la dreta. Si l'enter és negatiu s'imprimeix el signe `-` immediatament davant del primer dígit. Vegem-ne alguns exemples:

Valor en memòria	*Camp imprès* (*format* `i4`)
2537	2537
37	37
–19	–19
38529	****
–7514	****

- `fn.k`: s'utilitza per escriure dades reals i reals de precisió doble amb notació normal. El camp imprès es justifica per la dreta. `k` és un enter que indica el nombre de xifres decimals, de manera que tindrem, de dreta a esquerra, `k` xifres decimals, el punt decimal, les xifres significatives de la part entera (o un zero), el signe `-` si el valor és negatiu i, si calen, espais en blanc per completar el camp. El valor de `n` ha de ser $\geq$ `k+1` (es reserva una posició per al punt decimal) o $\geq$ `k+2` si la dada pot prendre valors negatius. Per exemple:

Valor en memòria	*Camp imprès* (*format* `f4.1`)
59	59.0
3,76	3.8
0,02	0.0
–5,14	–5.1
123,9	****
–78,5	****

- `en.k`: s'utilitza per escriure dades reals (de precisió senzilla) amb notació científica. El camp imprès es justifica per la dreta. `k` és un enter que indica el nombre de xifres decimals, de manera que tindrem, de dreta a esquerra, `Em` on `m` és l'exponent del factor 10^m representat amb dos dígits i el signe, `k` xifres decimals, el punt decimal, un zero com a part entera, el signe `-` si el valor és negatiu i, si calen, espais en blanc per completar el camp. El valor de `n` ha de ser $\geq$ `k+6` (6 posicions es reserven per a `Em`, el punt decimal i el zero) o $\geq$ `k+7` si el valor pot ser negatiu. Per exemple:

Valor en memòria	*Camp imprès* (*format* `e9.2`)
59	0.59E+02
0,02	0.20E–01
2000	0.20E+04
–0,1	–0.10E+00

- `dn.k`: s'utilitza per escriure dades reals de precisió doble amb notació científica. El compilador *g77* escriu el camp imprès com en el model `en.k`. Altres compiladors escriuen una `D` en comptes d'una `E` en la part de l'exponent de 10. Si l'exponent té tres dígits no s'escriu la lletra `E` o `D` . Per exemple:

Valor en memòria	*Camp imprès* (*format* `d9.2`)
59	0.59E+02 o 0.59D+02
–0,1	–0.10E+00 o –0.10D+00
$5,7 \times 10^{-112}$	0.57–112

- `gn.k`: s'utilitza per escriure dades reals i reals de precisió doble. Si el valor absolut de la dada està comprès entre 0.1 i $10^{\texttt{k}}$ el camp imprès s'escriu com en el model `fn.k` però deixant quatre espais en blanc a la dreta i, si no està comprès en aquell interval, s'escriu com en el model `en.k` o el `dn.k`.

- `a` i `an`: s'utilitzen per escriure dades literals; és a dir, cadenes de caràcters. En la forma `a` s'imprimeix la cadena de caràcters amb una longitud igual a la declarada en la instrucció `character`. Així, si hem fet la declaració `character*12 cadena` i fem escriure la variable `cadena` amb format `a` s'escriuran exactament 12 caràcters, tot afegint blancs si fos necessari. En la forma `an` només s'imprimeixen `n` caràcters (començant per l'esquerra). Així, si la variable `cadena` considerada abans conté la dada `margarida`, en escriure-la amb format `a5` s'escriurà `marga`; en canvi, amb format `a` s'escriuria `margarida` (amb tres espais en blanc al final).

- `nx`: s'utilitza per inserir `n` espais en blanc. Per exemple, el format

  ```
  125 format(f10.5,7x,f10.5)
  ```

  permetrà escriure dos dades reals (amb 10 posicions per a cada una d'elles) separades per set espais en blanc.

- `/`: s'utilitza per forçar un canvi de línia. Actua com a separador, de manera que no cal incloure comes abans i després de la barra. Per exemple, el format

  ```
  125 format("Temperatura =",f10.5/"Pressio =",f10.5)
  ```

  permetrà escriure dos dades reals en línies diferents.

Es pot afegir un nombre de repetició davant d'una especificació de format en comptes de repetir l'especificació; per exemple, la instrucció

```
190 format(f10.5,f10.5,f10.5,f10.5)
```

és equivalent a

```
190 format(4f10.5)
```

Si l'especificació és més complexa s'han de fer servir parèntesis; així,

```
150 format(f10.5,4x,f10.5,4x,f10.5,4x)
```

equival a

```
150 format(3(f10.5,4x))
```

Sempre que s'escriu una dada real amb menys xifres significatives que les emmagatzemades en la memòria s'arrodoneix el seu valor.

3.5 Instrucció *data*

La instrucció `data` permet assignar valors inicials a variables d'una manera alternativa a la indicada en la secció 2.5. La sintaxi de la versió més senzilla d'aquesta instrucció és

```
data llista de variables / llista de constants /
```

on la *llista de constants* conté els valors que s'han d'assignar a les variables de la *llista de variables*. Ambdues llistes han de contenir el mateix nombre d'elements, i el tipus de cada variable ha de ser el mateix que el de la constant corresponent.

Per exemple, la instrucció

```
data i, a, x0 / 3, 2.5, 1994.9 /
```

dóna els valors inicials 3, 2,5 i 1994,9 a les variables `i`, `a` i `x0`, respectivament.

Es pot evitar la repetició de constants idèntiques fent servir expressions del tipus `n*const`, on `n` és el nombre de vegades que s'ha de repetir la constant `const`. Així, la instrucció

```
data i, a, b, c, d / 1, 4*0.0 /
```

és equivalent a

```
data i, a, b, c, d / 1, 0.0, 0.0, 0.0, 0.0 /
```

La instrucció `data` també permet inicialitzar variables indexades. Per exemple, amb les instruccions:

```
dimension vec(20)
data vec / 20*0.0 /
```

els 20 components del vector `vec` reben valors inicials nuls. En el cas de les matrius l'assignació s'efectua per columnes; així, les instruccions

```
dimension mat(5,3)
data mat / 5*1, 10*0 /
```

assignen un 1 als elements de la primera columna i un 0 als restants.

L'assignació de valors de les instruccions `data` es fa durant la compilació, per la qual cosa no es poden fer servir instruccions d'aquest tipus per reassignar variables durant l'execució del programa. D'acord amb això, podem considerar-les com un cas especial d'instrucció de declaració, i és convenient posar-les a continuació de les altres instruccions de declaració del programa o subprograma.

3.6 Instrucció *common*

Hem vist que els paràmetres d'un subprograma permeten intercanviar informació amb el programa que l'ha cridat. Hi ha una manera alternativa d'assolir aquest objectiu, que consisteix a definir una zona de memòria comuna (és a dir, una col·lecció de variables) accessible des del programa principal i/o des de qualsevol subprograma. Per poder accedir-hi des d'un (sub)programa s'ha d'incloure una instrucció de declaració `common` en la zona de declaració corresponent:

```
common / nom / llista de variables
```

on *nom* és l'identificador de la zona de memòria comuna (pot haver-n'hi més d'una) i *llista de variables* és la relació de variables que constitueixen la zona.

Es pot definir una única zona comuna sense nom de la manera següent:

```
common llista de variables
```

La instrucció `common` ha d'anar després de les instruccions de declaració implícita i/o explícita. Com que els `common` defineixen zones de memòria, no podem incloure una mateixa variable en dos instruccions `common` diferents.

Com a exemple presentem un programa que calcula i compara els valors d'una paràbola en dos punts fent servir una funció externa d'una sola variable. Els paràmetres que defineixen la paràbola es passaran a la funció mitjançant una instrucció `common`:

```
      common a, b
      write (6,*) "Entreu els parametres de la parabola"
      read (5,*) a, b
      write (6,*) "Entreu els valors dels dos punts"
      read (5,*) x1, x2
      if (parab(x1) .ge. parab(x2)) then
        write (6,*) "El valor de la parabola es mes gran o igual en",
     $              " el primer punt"
      else
        write (6,*) "El valor de la parabola es mes petit en el",
     $              " primer punt"
      end if
      end
      function parab(x)
      common a, b
      parab = a*x**2 + b
      end
```

3.7 Instrucció *go to*

Aquesta instrucció s'explicarà perquè ha format part del llenguatge Fortran des dels seus inicis, i en les seves primeres versions s'usava força. No obstant això, ja hem indicat que, en general, no és recomanable utilitzar-la. La seva sintaxi és

```
      go to etiq
```

on *etiq* és una etiqueta que identifica una altra instrucció del programa (vegeu l'apartat 2.3.3), la qual pot estar abans o després de la instrucció `go to`. Després d'executar-se aquesta instrucció l'execució continua a partir de la instrucció que porta l'etiqueta; dit d'una altra manera, la instrucció `go to` transfereix el control de l'execució del programa a la instrucció etiquetada.

Per exemple, suposem que hem de calcular les mitjanes de moltes parelles de xifres amb el programa de la pàgina 22 i volem estalviar-nos el temps de teclejar cada vegada el nom del programa per executar-lo. La solució més ràpida és afegir, just abans de la instrucció `end`, una instrucció `go to` que transfereixi el control de l'execució a l'inici del programa, de manera que aquest torni a començar en arribar al final:

```
 75   write (6,*) "Teclegeu els nombres que voleu amitjanar, separats pe
     $r un espai en blanc o per una coma, i premeu Retorn"
      read (5,*) a, b
      c = (a+b)/2
      write (6,*) "La mitjana es:", c
      go to 75
      end
```

El problema és que aquest programa no s'aturaria, i hauríem d'avortar-lo amb la seqüència de tecles ctrl + C. Aquest problema es podria evitar afegint, a continuació de la lectura de les dades, una estructura alternativa que permeti aturar l'execució introduint unes dades determinades; per exemple, dos zeros:

```
 75   write (6,*) "Teclegeu els nombres que voleu amitjanar, separats pe
     $r un espai en blanc o per una coma, i premeu Retorn"
      write (6,*) "(per aturar el programa heu d'entrar dos zeros)"
      read (5,*) a, b
      if (a .eq. 0.0 .and. b .eq. 0.0) stop
      c = (a+b)/2
      write (6,*) "La mitjana es:", c
      go to 75
      end
```

però és preferible prescindir de les instruccions `go to` i `if` i utilitzar, al seu lloc, un bloc `do while` - `end do`:

```
      write (6,*) "Teclegeu els nombres que voleu amitjanar, separats pe
     $r un espai en blanc o per una coma, i premeu Retorn"
      read (5,*) a, b
      do while (a .ne. 0.0 .or. b .ne. 0.0)
        c = (a+b)/2
        write (6,*) "La mitjana es:", c
        write (6,*) "Teclegeu els nombres que voleu amitjanar, separats"
     $  ," per un espai en blanc o per una coma, i premeu Retorn"
        write (6,*) "(per aturar el programa heu d'entrar dos zeros)"
        read (5,*) a, b
      end do
      end
```

3.8 *Do* amb etiqueta

Existeix una forma alternativa del `do` amb índex que fa ús de les etiquetes esmentades en l'apartat 2.3.3:

```
      do etiq  index=valin,valfin,incr
        conjunt d'instruccions
 etiq continue
```

on *etiq* és una etiqueta que identifica la instrucció que tanca el conjunt d'instruccions a repetir. De fet, aquesta etiqueta es pot posar en qualsevol instrucció executable, però s'acostuma a utilitzar la instrucció `continue` —que no fa res d'especial— per tancar el bloc.

Així, el programa de l'exemple de la pàgina 45 es podria haver escrit de la manera següent:

```
      do 27 i = 1, 9
        do 86 j = 1, 9
          write (6,*) i, j
 86     continue
 27   continue
      end
```

Com que un bloc `do` - *etiqueta* intern ha d'estar contingut completament dins d'un altre extern, és evident que l'etiqueta de l'intern ha d'estar sempre abans que la de l'extern. Observeu que no cal que les etiquetes estiguin ordenades.

El `do` amb etiqueta és, de fet, la forma estàndard del `do` amb índex en les especificacions del Fortran 77, però quasi tots els compiladors accepten la forma, més elegant, que acaba amb l'`end do`.

3.9 Altres exercicis

Exercici 3.27
Diuen que el joc d'escacs el va crear un súbdit per entretenir el seu ociós rei, i aquest, per agrair-li l'invent, li va dir que demanés algun obsequi. Ell li va demanar com a recompensa un gra de blat per la primera casella, dos per la segona, quatre per la tercera, vuit per la quarta, i així successivament fins a cobrir amb aquesta progressió geomètrica les 64 caselles que formen el tauler. Feu un programa que calculi, amb precisió doble, el nombre de grans de blat que demanava el súbdit. *El resultat exacte és*: 18.446.744.073.709.551.615.

Exercici 3.28
Feu un programa que demani a l'usuari un nombre natural n i un de real a_1 i escrigui els n primers termes de la successió definida per la relació de recurrència $a_{i+1} = a_i^2$. *Comprovació* : Per a $n = 4$ i $a_1 = 3$, la successió és 3, 9, 81 i 6561.

Exercici 3.29
Feu un programa que determini el nombre enter n més petit que compleixi $(n + 2)/(2n)! < \varepsilon$, on ε és un valor a introduir per l'usuari. Empreu una funció externa que calculi el factorial d'un nombre enter i retorni el resultat en una variable real de precisió doble. *Comprovació*: Per a $\varepsilon = 10^{-20}$ heu d'obtenir $n = 12$.

Exercici 3.30
Volem fer un programa que digui a l'usuari si un nombre natural, entrat com a dada, és capicua o no. Una opció és demanar a l'usuari quants dígits té el nombre (amb un màxim de, per exemple, 100) i fer-li entrar els dígits d'un en un per poder guardar cada un en un component d'un vector. *Suggeriment*: Podeu pensar com ha de ser l'algorisme quan el nombre de dígits és parell i, després, veure si funciona per a nombres senars de dígits.

Exercici 3.31
Feu un programa que escrigui totes les combinacions (sense repetició) de les cinc vocals agafades de 3 en 3 (aei, aeo, aeu, aio, ..., eou, iou).

Exercici 3.32
El compilador g77 incorpora una funció intrínseca, `rand(llavor)`, que genera un nombre real pseudoaleatori entre 0 i 1 a partir d'un nombre natural que s'anomena la llavor.
a) Feu un programa que demani un nombre natural (la llavor), generi a partir d'aquest un nombre pseudoaleatori entre 0 i 1, el multipliqui per 100 i trunqui els decimals del valor obtingut, de manera que en resulti un nombre enter pseudoaleatori comprès entre 0 i 100. Després, el programa ha d'anar demanant a l'usuari nombres naturals entre 0 i 100 fins que encerti el pseudoaleatori. Per tal de facilitar-li la feina, el programa indicarà si cada nombre entrat és major o menor que el que ha d'encertar. Quan l'encerti, el programa li dirà quants intents li han calgut per fer-ho.
b) Modifiqueu el programa anterior perquè el nombre pseudoaleatori generat arribi fins a 1000 i perquè doni més pistes a l'usuari per tal de facilitar-li la feina: "vas molt desencaminat", "estàs a prop", "ai, que et cremes!", etc.

Exercici 3.33
El procediment de D'Hondt (Víctor D'Hondt, 1878) per repartir els n escons de cada circumscripció d'un parlament d'acord amb els nv_i vots de cada partit funciona de la manera següent: els vots rebuts per cada partit es divideixen per 1, per 2, ..., per $n-1$ i per n. Si hi ha m partits això produeix una matriu de m files (els partits) i n columnes (els divisors). De l'article 163 de la *Ley Orgánica 5/1985, de 19 de junio, del Régimen General Electoral* es desprèn que els decimals d'aquests quocients s'han de truncar; és a dir, el que s'ha de guardar a la matriu és la part entera de cada quocient. El primer escó s'assigna al partit al qual correspon l'element màs gran de la matriu. Aquest element es posa a zero i s'assigna el segon escó al partit que té l'element més gran dels que queden, i així successivament fins que s'hagin assignat tots els escons de la circumscripció.
a) Feu un programa que demani els nombres d'escons i de partits d'una circumscripció, i els vots rebuts per aquests. El programa ha de determinar el nombre d'escons que correspon a cada partit. *Comprovació*: Per a 6 partits amb nombres de vots 168000, 104000, 72000, 64000, 40000 i 32000, i 8 escons a repartir, s'han d'obtenir els nombres d'escons següents: 4, 2, 1, 1, 0 i 0.
b) Modifiqueu el programa de l'apartat anterior perquè demani també el nom de cada partit i escrigui la llista de partits amb el nombre d'escons assignat a cadascun.

Exercici 3.34
El procediment de D'Hondt modificat afegeix el requisit següent: només els partits amb un percentatge mínim de vots poden tenir representació parlamentària. Per aplicar-lo es prescindeix dels partits que no arribin a aquest mínim i s'aplica el procediment de D'Hondt als restants. Modifiqueu el programa de l'exercici anterior per tenir en compte aquesta modificació. *Comprovació*: Per a 6 partits amb nombres de vots 665000, 621000, 281000, 248000, 231000 i 78000, i 85 escons a repartir, amb un percentatge mínim del 5%, s'han d'obtenir els nombres d'escons següents: 28, 26, 12, 10, 9 i 0. Calculeu l'assignació d'escons corresponent a aquests mateixos nombres de vots si no hi hagués percentatge mínim.

Exercici 3.35
Suposem que en un parlament hi ha np partits representats. Feu un programa que demani a l'usuari el nombre de partits amb representació parlamentària (amb un màxim de 31) i el nombre d'escons i el nom de cada partit, i determini quins partits individuals o coalicions poden tenir majoria absoluta; és a dir, més de la meitat del nombre total d'escons. El programa ha d'escriure, per cadascun d'aquests, el nombre total d'escons juntament amb els noms dels partits integrants. *Suggeriment* : Transformeu l'exercici 2.67 en una subrutina que passi a format binari un nombre enter expressat en format decimal, tot retornant els dígits binaris en els elements d'un vector enter de 31 components.

Utilitzeu aquesta subrutina per generar els binaris corresponents als enters 1, 2, ..., $2^{np}-1$. Per a cadascun d'aquests calculeu la suma dels productes dels seus dígits binaris pels nombres d'escons de cada partit, tot començant per la dreta; per exemple, per al nombre binari 1...010 s'ha de calcular la suma 0 × (nombre d'escons del primer partit) + 1 × (nombre d'escons del segon partit) + 0 × (nombre d'escons del tercer partit) + ... + 1 × (nombre d'escons de l'últim partit). Si aquest resultat és més gran que la meitat del nombre total d'escons, la coalició formada pels partits que han contribuït a la suma tindrà majoria absoluta. *Comprovació*: Per als resultats de les eleccions autonòmiques del 2006 al Parlament de Catalunya (Ciutadans: 3, CiU: 48, ICV: 12, ERC: 21, PP: 14 i PSC: 37), s'obtenen 32 coalicions possibles (una altra cosa és que s'entenguin entre ells).

Exercici 3.36
Quan un escalador assegurat amb corda cau, la pèrdua d'energia potencial gravitatòria de l'escalador en el moment en què s'atura (ΔE_{pot}) ha de ser igual a l'energia de tensió acumulada per la corda en estirar-se (ΔE_{tens}) més l'energia dissipada per fregament (E_{freg}) (vegeu la figura 3.3):

$$\Delta E_{pot} = \Delta E_{tens} + E_{freg}$$

Si suposem que la corda es comporta de manera perfectament elàstica amb una constant de força k ($\Delta E_{tens} = \frac{1}{2}k(\Delta L)^2$, on ΔL és l'allargament de la corda), és fàcil comprovar que l'equació anterior es pot expressar de la manera següent:

$$mg(f+a) = \frac{1}{2}ca^2 + \frac{E_{freg}}{L} \tag{3.5}$$

on m és la massa de l'escalador que cau, $g = 9,8$ m/s és l'acceleració de la gravetat, L és la longitud de corda que hi ha entre l'escalador i l'assegurador (la *corda activa*), f (*factor de caiguda*) és el quocient entre la distància vertical recorreguda per l'escalador durant la caiguda fins que es comença a tensar la corda (d) i la longitud de corda activa ($f = d/L$), $a = \Delta L/L$ és l'allargament relatiu que experimenta la corda en el moment d'estirament màxim, i $c = kL$ és un coeficient que caracteritza la rigidesa de la corda.
Les especificacions tècniques d'una corda solen incloure l'allargament relatiu a_e i la tensió màxima $T_e = k\Delta L = ca_e$ que experimenta en una caiguda de factor $f_e = 1{,}77$ d'un escalador de $m_e = 80$ kg (l'anomenarem *caiguda estàndard*). A partir d'aquestes dades podem determinar el coeficient de rigidesa $c = T_e/a_e$ i l'energia de fregament dissipada per unitat de longitud de corda activa ($E_{freg}/L = m_e g(f_e + a_e) - ca_e^2/2$). Si anomenem r la relació entre l'energia de fregament i l'energia potencial perduda:

$$r = \frac{E_{freg}}{\Delta E_{pot}} = 1 - \frac{ca_e^2}{2m_e g(f_e + a_e)}$$

podem expressar el balanç energètic (3.5) de la manera següent:

$$(1-r)mg(f+a) = \frac{1}{2}ca^2 \tag{3.6}$$

Si suposem que r es manté constant en diferents caigudes amb la mateixa corda, podem utilitzar els valors de c i r determinats a partir de la caiguda estàndard per calcular l'allargament relatiu a (equació 3.6) i la tensió màxima $T = ca$ en qualsevol altra caiguda, caracteritzada per m i f.
a) Feu un programa que rebi com a dades les especificacions d'una corda (m_e, f_e, a_e i T_e) i calculi c i r. Després el programa demanarà els valors de m i f per a una caiguda qualsevol i calcularà l'allargament relatiu a i la tensió màxima T corresponents. *Comprovació*: Les especificacions d'una corda indiquen que s'allarga un 38% ($a_e = 0,38$) i pateix una tensió màxima de 7400 N en una caiguda de factor 1,77 d'un escalador de 80 kg. Per a una caiguda de factor $f = 1,5$ d'un escalador de 55 kg assegurat amb aquesta corda heu d'obtenir $a = 0,287$ i $T = 5594$ N (aquesta tensió, que determina la intensitat de l'estrebada que rep l'escalador, és més de 10 vegades el seu pes!).
b) Modifiqueu el programa anterior perquè escrigui una taula amb els allargaments relatius i una altra amb les tensions màximes que pateix la corda per a caigudes de factors 0,5, 1,0, 1,5 i 2,0 amb masses entre 50 i 100 kg incrementades de 5 en 5 kg.

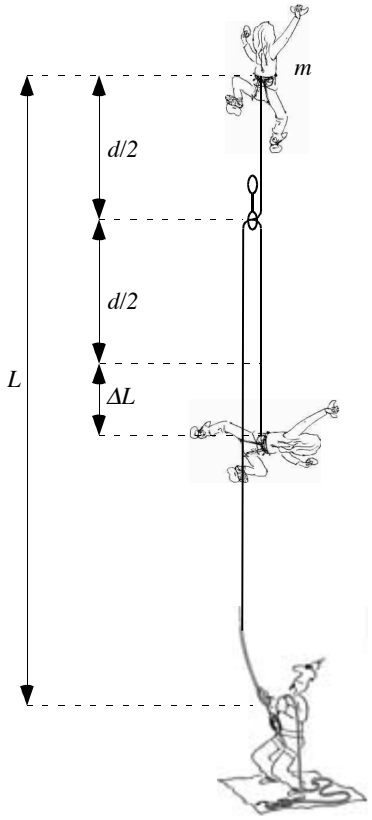

Figura 3.3: Esquema d'una caiguda d'un escalador.

Exercici 3.37
Feu un programa que avaluï el sinus d'un angle x en radians fent servir una funció externa de doble precisió que sumi els termes de la sèrie

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

que siguin més grans en valor absolut que un paràmetre petit ε, subministrat com a dada. El programa ha de calcular també el valor del sinus mitjançant la funció incorporada `sin` i escriure els dos resultats per poder-los comparar (proveu amb diferents valors de ε des de 10^{-7} fins a 10^{-15}). *Suggeriment*: Tingueu en compte que $\frac{x^n}{n!} = \frac{x^{n-2}}{(n-2)!}\frac{x^2}{n(n-1)}$.

Exercici 3.38
Un compost químic A reacciona per donar un producte C segons el mecanisme de reacció següent:

$$A \xrightarrow{k_1} B$$
$$A + B \underset{k_3}{\overset{k_2}{\rightleftarrows}} C$$

on k_1, k_2 i k_3 són les constants de velocitat dels processos elementals indicats (totes tres són ≥ 0). Si anomenem x, y i z les concentracions de les espècies A, B i C, la seva variació amb el temps està determinada per les equacions diferencials acoblades següents:

$$\frac{dx}{dt} = -k_1 x - k_2 xy + k_3 z \qquad \frac{dy}{dt} = k_1 x - k_2 xy + k_3 z \qquad \frac{dz}{dt} = k_2 xy - k_3 z$$

Aquest sistema es pot substituir per un sistema amb diferències finites:

$$\begin{aligned} \frac{x(t+\Delta t) - x(t)}{\Delta t} &= -k_1 x - k_2 xy + k_3 z \\ \frac{y(t+\Delta t) - y(t)}{\Delta t} &= k_1 x - k_2 xy + k_3 z \\ \frac{z(t+\Delta t) - z(t)}{\Delta t} &= k_2 xy - k_3 z \end{aligned}$$

la qual cosa introduirà un error més petit com més petit sigui l'interval Δt. Aquest sistema permet calcular una sèrie de ternes de valors $x(t)$, $y(t)$, $z(t)$, on cada terna s'obté de l'anterior mitjançant les expressions:

$$\begin{aligned} x(t+\Delta t) &= [-k_1 x - k_2 xy + k_3 z]\Delta t + x(t) \\ y(t+\Delta t) &= [k_1 x - k_2 xy + k_3 z]\Delta t + y(t) \\ z(t+\Delta t) &= [k_2 xy - k_3 z]\Delta t + z(t) \end{aligned}$$

a) Feu un programa que demani a l'usuari les constants k_1, k_2 i k_3, uns valors inicials de les concentracions $x(0)$, $y(0)$ i $z(0)$, un temps final t_f i el nombre n de valors equidistants de t ($t_1 = \Delta t$, $t_2 = 2\Delta t$, ..., $t_n = n\Delta t = t_f$), en què volem que calculi els valors de $x(t)$, de $y(t)$ i de $z(t)$. *Comprovació*: Per a $k_1 = 0,2$, $k_2 = 6,0$ i $k_3 = 0,1$ amb concentracions inicials $x(0) = 0,3$, $y(0) = 0$ i $z(0) = 0$ (en unitats adients), les concentracions a un temps $t_f = 3$ són, aproximadament, 0,1073, 0,0385 i 0,0771.
b) Modifiqueu el programa anterior perquè demani a l'usuari si vol una taula amb els successius conjunts de valors t_i, $x(t_i)$, $y(t_i)$, $z(t_i)$ per a $i = 0$, 1, ..., n. Aquesta taula s'haurà d'escriure en el monitor i en un fitxer amb format CSV.
c) Llegiu la taula generada en l'apartat anterior amb l'OpenOffice (o alguna aplicació similar) i representeu gràficament les tres corbes concentració/temps.

Exercici 3.39
La mecànica quàntica prediu que, si mesuréssim la posició de l'electró relativa al nucli per a molts àtoms d'hidrogen, tots en el seu estat de mínima energia ($1s$), el 90% de les vegades la posició obtinguda estaria dins una esfera de radi R, on R, expressat en bohrs (1 bohr $= 5,29177 \times 10^{-11}$m), és la solució de l'equació (vegeu l'exercici 3.11, pàgina 83; la unitat "bohr" és igual a a_0)

$$2R = \ln\left(20R^2 + 20R + 10\right)$$

Feu un programa que calculi el valor de R amb una precisió ε per aproximacions successives a partir d'un valor de prova inicial. *Resultat*: 2,66116 bohrs.

Exercici 3.40
a) Escriviu una funció externa que avaluï la funció error (la qual apareix, per exemple, en el tractament estadístic de mesures experimentals):

$$f_{err}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du$$

per a un valor positiu de x introduït per l'usuari resolent numèricament la integral que intervé a la seva definició mitjançant algun dels mètodes descrits en la secció 3.2 (vegeu l'exercici 3.7, pàgina 82). Preneu un nombre de subdivisions (nsu) de l'interval d'integració ($u \in [0, x]$) proporcional a la longitud d'aquest —de manera que la longitud de les subdivisions sigui sempre aproximadament la mateixa—; per exemple, podeu prendre $nsu \approx 10^4 x$ (subdivisions de longitud $h \approx 10^{-4}$) amb un mínim de 10, i fer que el subprograma calculi un valor acurat de h com x/nsu. Afegiu a la funció externa un programa principal que la cridi per calcular la funció error en un punt x demanat a l'usuari i escrigui el resultat. *Suggeriment*: Convé utilitzar variables de precisió doble (π = 3,1415926535897932...). *Comprovació*: $f_{err}(1) = 0{,}842701$.
b) Escriviu un programa que faci servir la funció externa demanada en l'apartat *a)* —sense modificar-la— per avaluar la funció error en 200 punts equidistants de l'interval (0,2] —en què el primer és el 0,01— i escrigui una taula amb les 200 parelles de valors $x, f_{err}(x)$ en el monitor i en un fitxer amb format CSV.
c) Llegiu la taula generada en l'apartat anterior amb l'OpenOffice (o alguna aplicació similar) i representeu gràficament la funció error en l'interval (0,2].

Exercici 3.41
Feu un programa que calculi numèricament la integral de la funció error entre dos límits a i b especificats per l'usuari ($\int_a^b f_{err}(x)dx$), amb $0 \leq a < b$, tot utilitzant la funció externa demanada en l'apartat *a)* de l'exercici anterior per avaluar l'integrand. Preneu $nsx \approx 5000(b-a)$ subdivisions de l'interval d'integració (la longitud aproximada del qual serà 1/5000), amb un mínim de 10. *Comprovació*: Per a $a = 0$ i $b = 1$ s'obté 0,486061.

Exercici 3.42
a) Una manera alternativa d'expressar la funció error per a un valor positiu de x és:

$$f_{err}(x) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{n!(2n+1)} = \frac{2}{\sqrt{\pi}} \left(x - \frac{x^3}{3} + \frac{x^5}{10} - \ldots \right)$$

Escriviu una funció externa que avaluï la funció error per a un valor de x introduït com a dada sumant els termes de l'expressió anterior majors que un valor ε demanat a l'usuari. *Suggeriment*: Tingueu en compte que cada terme es pot obtenir a partir de l'anterior multiplicant-lo per $\frac{-x^2(2n-1)}{n(2n+1)}$.
Comprovació: $f_{err}(1) = 0{,}842701$.
b) La funció error pren valors ≤ 1 per a qualsevol valor de x i tendeix cap a 1 quan $x \to \infty$. Feu un programa que utilitzi la funció externa de l'apartat *a)* i la de l'exercici 3.40 per avaluar la funció en els valors de x següents: 6, 8, 10, 12, 14... Comproveu que la primera funció produeix resultats incorrectes per als valors més grans (fins i tot amb variables de precisió doble). Això fa palesa la cura que s'ha de tenir amb els càlculs numèrics, sobretot quan impliquen diferències petites entre valors grans. Per tal de comprovar que aquesta situació es produeix en la funció de l'apartat *a)*, modifiqueu-la perquè l'usuari pugui fer que s'escrigui cada terme de la sèrie.

Exercici 3.43
Si suposem que l'energia interna d'un sòlid cristal·lí ve determinada pel moviment vibracional dels seus àtoms o molècules al voltant de les respectives posicions d'equilibri, i tenim en compte que aquestes vibracions no són completament independents entre elles —model de Debye—, obtenim l'expressió següent per a la capacitat calorífica molar a volum constant (C_V) en funció de la temperatura absoluta (T):

$$C_V = 9Rb^{-3} \int_0^b \frac{x^4 e^x}{(e^x - 1)^2} dx$$

on $b = \Theta_D/T$, essent Θ_D la temperatura de Debye del sòlid (relacionada amb la freqüència vibracional mitjana dels seus àtoms o molècules), i $R = 8{,}314472$ J K^{-1}mol^{-1} és la constant dels gasos. En aquest exercici utilitzarem aquesta expressió per calcular la capacitat calorífica d'un sòlid a diferents temperatures i compararem els valors calculats amb resultats experimentals.

a) Feu un programa que demani a l'usuari la temperatura de Debye del sòlid (Θ_D) i la temperatura (T) a la qual es vol calcular C_V (ambdues expressades en kelvins) i cridi a una funció externa que calculi $b = \Theta_D/T$ i C_V resolent numèricament la integral corresponent mitjançant el mètode dels rectangles (vegeu l'exercici 3.7, pàgina 82). Després, el programa principal ha d'escriure els valors de T i de C_V. *Suggeriment*: Per tal d'evitar la indeterminació del tipus 0/0 que presenta l'integrand en $x = 0$ preneu com a altura de cada rectangle el valor de l'integrand al costat dret del rectangle (equació (3.1)). Feu servir variables reals de precisió doble. Preneu com a nombre de subdivisions de l'interval d'integració $nsx \approx 5000b$ per tal que la longitud h de les subdivisions sigui aproximadament 1/5000 independentment del valor que prengui b (la funció ha de calcular el valor exacte de h). *Comprovació*: El resultat que s'obté per a la plata ($\Theta_D = 215$ K) a 5 K és $C_V = 0{,}0244478$ JK^{-1}mol^{-1}.

b) Fet un programa que faci servir la funció externa que heu escrit en l'apartat anterior (sense modificar-la) per obtenir una taula amb parelles de valors (T, C_V) per a la plata a temperatures entre 5 K i 210 K amb increments d'un kelvin. Aquesta taula s'ha d'escriure en un fitxer amb format CSV.

c) Mesures experimentals han produït els valors següents (T en K i C_V en JK^{-1}mol^{-1}):

T	C_V	T	C_V	T	C_V
5	0,0213	20,00	1,6715	83,91	18,10
6	0,0373	28,56	4,297	103,20	20,07
7	0,0632	36,16	7,088	124,20	21,27
8	0,0987	47,09	10,803	144,38	22,48
10	0,199	55,88	13,33	166,78	22,86
12	0,347	65,19	15,37	190,17	23,34
14	0,5590	74,56	16,90	205,30	23,45
16	0,8452				

Introduiu aquesta taula de valors en un fitxer amb format CSV i llegiu-la, juntament amb la que heu generat en l'apartat *b)*, amb l'OpenOffice (o alguna aplicació similar). Representeu la corba $C_V(T)$ experimental i la calculada.

d) Feu un programa que calculi l'increment d'energia interna molar que experimenta la plata quan s'escalfa a volum constant des d'una temperatura T_1 fins a una altra T_2 emprant la funció externa $C_V(\Theta_D, T)$ que heu fet en l'apartat *a)* per resoldre numèricament la integral:

$$\Delta U = \int_{T_1}^{T_2} C_V(T)dT$$

essent T_1 i T_2 dades a introduir per l'usuari. Preneu com a nombre de subdivisions de l'interval d'integració $nsT = 50|T_2 - T_1|$, la qual cosa fa que cada subdivisió sigui, aproximadament, de 0,02 K. *Comprovació*: Per a $T_1 = 200$ K i $T_2 = 250$ K s'obté $\Delta U = 1191{,}76$ J.

APÈNDIXS

A Evolució dels sistemes operatius

Les dades contingudes en aquest apèndix han estat extretes de pàgines web molt diverses; tot i que hem procurat escollir fonts que ens inspirin confiança, no podem garantir la seva completa fiabilitat.

Els primers ordinadors (mitjan 1940 fins a mitjan 1950) no disposaven de sistema operatiu pròpiament dit: el maquinari es manipulava de manera molt directa mitjançant commutadors elèctrics i, com a molt, un rudimentari teclat hexadecimal (amb poc més de 16 tecles: 0, 1, 2, ..., 9, a, b, ..., f), el seu estat es detectava a través d'indicadors lluminosos, i els programes es carregaven manualment en l'ordinador. Aviat es va veure que aquesta manera de treballar era lenta i poc flexible, i que calia simplificar i automatitzar el procediment per tal d'optimitzar el rendiment del caríssim maquinari. En particular, es començaren a desenvolupar dispositius d'entrada i sortida més sofisticats (teclats alfanumèrics, lectores de cintes de paper i targetes perforades, pantalles amb una línia de text, etc.) i sistemes d'*execució per lots* (*cues de batch*): en lloc de manipular directament l'ordinador, els programadors entregaven els seus programes a un operador que els disposava de manera que, en acabar l'execució d'un, comencés automàticament la d'un altre. Això exigia afegir a cada programa instruccions per reservar la quantitat de memòria necessària, carregar-lo en memòria, etc. Estaven naixent els sistemes operatius. Per aprofitar les estones en què l'ordinador "perdia" el seu valuós temps esperant que s'efectués una entrada o una sortida d'informació, es desenvoluparen, cap a mitjan 1960, sistemes *multiprogramació* o *multitasca*, els quals permetien carregar en memòria dos o més programes simultàniament i avançar en l'execució d'un d'ells mentre un altre esperava que es completés una operació d'entrada o sortida. Es deia també que la CPU estava *multiplexada* (compartida) entre diferents treballs.

Els primers sistemes operatius només permetien treballar en mode text; és a dir, sense interfície gràfica. Entre aquests pioners cal destacar, per la seva versatilitat i la influència que ha tingut en els sistemes actuals, l'Unix. Desenvolupat originalment el 1969 per Ken Thompson i altres de l'empresa AT&T amb el nom d'UNICS,[1] aquest sistema fou cedit a la Universitat de Berkeley el 1978 perquè el distribuís de manera gratuïta (BSD: Berkeley Software Distribution). Actualment hi ha nombroses derivacions,[2] tant gratuïtes —com el GNU/Linux— com de pagament —com el Mac OS X— que incorporen interfícies gràfiques per tal de facilitar-ne la utilització.

L'origen de les interfícies gràfiques (GUI) se situa el 1973, quan la casa Xerox desenvolupà l'Alto. Aquest prototip fou el primer ordinador amb una GUI. A més, el monitor era d'una grandària inusual en aquell temps (mida foli). També presentava altres innovacions revolucionàries, com la xarxa Ethernet, el correu electrònic, la impressora làser, un editor de text adaptat a la GUI, etc. El primer ordinador comercial amb GUI fou el Xerox Star 8010 o Dandelion, una evolució de l'Alto que sortí al mercat el 27 d'abril de 1981. Aquesta màquina no va tenir èxit comercial a causa, sobretot, de

[1] L'UNICS és una derivació del MULTICS (MULtiplexed Information & Computing Service), sistema *multitasca* desenvolupat pels Laboratoris Bell (AT&T), GE (General Electric) i el MIT (Massachusets Institute of Technology) a partir del 1965 amb la idea de permetre a una gran comunitat d'usuaris treballar simultàniament amb un mateix ordinador des de diferents terminals (sistema *multiusuari*). El MULTICS, al seu torn, derivava del CTSS (Compatible Time-Sharing System), desenvolupat al MIT entorn de 1962 per un equip dirigit per Fernando José Corbató, que es pot considerar el primer sistema multitasca. L'UNICS fou, originalment, un sistema monousuari, és a dir, no admetia usuaris simultanis (un dels enginyers l'anomenà, en to de broma, UNIplexed Information & Computing Service). El 1973 derivà en l'Unix (Ken Thompson i Dennis Ritchie), que és un sistema multitasca i multiusuari.

[2] La pàgina web *www.levenez.com/unix/* presenta un detallat esquema d'aquestes derivacions.

dos motius: el seu cost elevat (16595 $) no el feia apte per a l'usuari domèstic, i la relativa lentitud de resposta de la GUI exasperava els informàtics professionals. El 1979 Xerox havia fet una presentació de l'Alto a la qual havien assistit, entre d'altres, Bill Gates, cofundador d'una petita empresa de programari anomenada Microsoft, i Steve Jobs, cofundador d'Apple Computer. Aquesta darrera empresa havia fet una important contribució a la popularització de la informàtica amb l'Apple II (abril de 1977), un ordinador assequible (1295 $), versàtil (monitor en colors amb capacitat gràfica, so, llenguatge BASIC incorporat a la ROM, etc.) i amb una difusió prou gran per poder ser considerat el primer ordinador domèstic o personal. Sembla que Gates no donà gaire importància a les innovacions de l'Alto, però Jobs en quedà fascinat —sobretot per la interfície gràfica— i, convençut que produirien un pas de gegant en l'extensió de la informàtica en l'àmbit domèstic, decidí reorientar el disseny del Lisa, un potent ordinador personal que estava desenvolupant, per incorporar-hi una GUI juntament amb altres innovacions. El Lisa sortí al mercat el gener de 1983, però, conscient que el cost de la nova màquina (9995 $) era encara excessiu (s'havien invertit 50 milions de dòlars i 200 anys×persona), Jobs havia engegat un projecte més modest i assequible —el Macintosh—, que començà a vendre's per 2495 $ al gener de 1984. Aquest fou el primer ordinador amb sistema operatiu basat en una GUI (el Mac OS) que triomfà comercialment. Durant els anys següents, la major part dels sistemes operatius han anat incorporant GUI molt semblants, la qual cosa ha facilitat molt la seva utilització i, juntament amb la baixada de preu dels ordinadors personals, han estat fets clau en la massiva popularització d'aquestes màquines.

A la vista del creixent volum de negoci que representava la informàtica personal, la prestigiosa casa IBM havia decidit fer una aposta seriosa per aquest sector llançant al mercat el 5150 PC el setembre de 1981, un ordinador força senzill (monitor monocrom no gràfic, ROM molt bàsica, etc.) però bastant assequible (3000 $ sense sistema operatiu). Aquest ordinador havia de funcionar amb una versió adaptada a processadors de 16 bits del sistema operatiu (de mode text) CP/M (Control Program for Microcomputers), desenvolupat el 1974 per Gary Kildall, fundador de Digital Research Incorporated (DRI), però, a última hora, IBM trencà les relacions amb DRI i optà per encarregar un sistema operatiu a Microsoft. Aquesta empresa no tenia el producte que IBM demanava, però Bill Gates sabia que Seattle Computing Products (SCP) disposava d'un sistema que satisfeia les necessitats d'IBM —el QDOS (Quick and Dirty Operating System)— i acceptà l'encàrrec. De seguida comprà aquest sistema (sembla que per 20000 $) i n'oferí una versió revisada a IBM amb el nom de PC-DOS (més tard MS-DOS: MicroSoft Disc Operating System). El QDOS era molt semblant al CP/M i, de fet, sembla que SCP havia decidit pel seu compte adaptar aquest últim al processador Intel 8086 dels seus ordinadors (un processador similar a l'Intel 8088 que incorporarien els PC d'IBM). A les proves del PC-DOS, IBM detectà errors i l'hagué de reescriure en part, la qual cosa va endarrerir la sortida al mercat de l'IBM PC-DOS 1.0, i els primers 5150 PC es vengueren sense sistema operatiu.[3] Tot i això, la reputació de la marca IBM va fer que el nou PC tingués un èxit comercial sense precedents.

L'excel·lent acollida que tingué al mercat el Macintosh va fer canviar l'estratègia de Microsoft, que, el novembre de 1985, afegí al DOS una aplicació que permetia efectuar les operacions més usuals amb una rudimentària GUI: el Windows 1.0. Aquest mateix any DRI, que havia perdut la batalla dels sistemes operatius de mode text (el CP/M havia quedat en desús i el posterior DR-DOS, tot i oferir notables avantatges sobre el MS-DOS, no havia pogut competir-hi), llançava el GEM (Graphic Environment Manager), una GUI per als PC molt similar a la desenvolupada per Apple i força més elaborada que el Windows 1.0 (suportava molts monitors, acceptava diversos llenguatges de programació, etc.), però tampoc en aquest terreny no va poder competir amb l'empresa que començava a dominar el mercat del programari. Amb els anys, les successives versions del Windows han anat superant les principals deficiències del Windows 1.0 i incorporant prestacions d'altres GUI més sofisticades (sobretot la del Mac OS), fins a arribar a imposar-se com el sistema operatiu més utilitzat arreu del món.

L'evolució dels sistemes operatius arrossegà, al seu torn, la del programari d'aplicació. El PC-DOS era un sistema amb seriosos inconvenients per adaptar-se a l'evolució del maquinari. Entre d'altres, era monotasca, i el processador 80286 que incorporà l'IBM PC-AT el 1984 havia estat dissenyat per Intel amb suport multitasca, per la qual cosa IBM va prometre als desenvolupadors d'aplicacions que

[3]De fet, IBM anuncià que la seva màquina podria funcionar amb tres sistemes operatius: l'IBM PC-DOS i el CP/M 86 de DRI, que encara no estaven a punt, i una versió de l'UCSD p-System (University of California at Sant Diego Pseudo code System). L'IBM PC-DOS, comercialitzat a un preu molt inferior al dels altres, fou pràcticament l'únic que es va vendre.

els proporcionaria un sistema operatiu multitasca. Amb aquesta fi havia engegat una col·laboració amb Microsoft per dissenyar un nou sistema operatiu, l'OS/2. IBM s'ocuparia principalment del nucli del sistema, i Microsoft, de la GUI —la qual cosa li proporcionaria una experiència molt útil de cara a la posada a punt del Windows, en la qual també treballava. Sembla ser que aquesta empresa va anar disminuint els recursos dedicats l'OS/2, fins que IBM decidí continuar en solitari amb el projecte. Simultàniament, Microsoft impulsava noves versions del Windows (el 3.0 sortiria el 1990) i reorientava la seva aportació a l'OS/2 cap al que anomenaria Windows NT (New Technology). L'OS/2, disponible al mercat des del 1987, va ser un fracàs comercial, la qual cosa provocà importants pèrdues als desenvolupadors que havien invertit esforços per emmotllar-se al nou sistema i no van tenir prou temps per adaptar les seves aplicacions al Windows 3.0 quan aquest sortí al mercat. D'altra banda, Microsoft havia estat treballant en una col·lecció d'aplicacions per a DOS/Windows: l'editor de textos Word, el full de càlcul Excel, el programa de presentacions PowerPoint, la base de dades Access, etc. Molts desenvolupadors anaren enfonsant-se alhora que Microsoft s'imposava, també, en el camp del programari d'aplicació.

Amb els anys, el Windows 3.0 ha evolucionat cap al Windows 95, el Windows 98 i el Windows ME. Paral·lelament el Windows NT donà lloc al Windows 2000, succeït pel Windows XP i, més tard, pel Windows Vista. L'altra línia de desenvolupament s'extingí amb la versió ME.

B Programari lliure

Als inicis de la informàtica —anys cinquanta, seixanta i setanta— la major part del programari es desenvolupava en universitats i centres de recerca. Quan un investigador creava una aplicació per resoldre un problema, la lliure distribució d'aquesta entre la gent interessada facilitava la verificació (i millora) del programari i dels algorismes en què es basava, alhora que donava prestigi a l'autor. A partir dels vuitanta aquesta tendència es va invertir: la major part del programari passà a desenvolupar-se en empreses sota uns drets d'autor que prohibeixen copiar-lo i/o modificar-lo (*programari propietari*). Si l'usuari d'aquest programari detecta un error no el pot solucionar, ja que, d'una banda, no se sol disposar del codi font i, de l'altra, no té el dret legal de fer-ho. L'única opció permesa és esperar que el fabricant el corregeixi i, possiblement, tornar a pagar per obtenir la versió corregida. Això pot tenir una altra conseqüència encara més greu: certs documents generats amb programari propietari només es poden llegir i modificar amb aquell programari (o amb un nombre reduït d'aplicacions), de manera que, si el fabricant deixa de mantenir-lo per adaptar-lo a noves màquines o sistemes operatius —el manteniment es fa sovint d'acord amb criteris de rendibilitat comercial—, és possible que no puguem recuperar la informació que contenen. El mateix pot passar si un document s'espatlla (es *corromp*) com a conseqüència d'un error en el programari que l'ha creat o en el sistema operatiu, d'una fallada del maquinari o d'un tall de corrent. Alguns desenvolupadors de programari s'han rebel·lat contra aquesta tendència i han impulsat un moviment que està guanyant força cada dia. Entre aquests cal destacar Richard Stallman, informàtic del Laboratori d'Intel·ligència Artificial del MIT (Massachusetts Institut of Technology) des de 1971, que el gener de 1984 renuncià al seu treball en aquest centre per poder dedicar-se al desenvolupament de *programari lliure* sense que ningú no pogués imposar-li condicions de distribució. D'acord amb Stallman, diem que un programari és lliure quan compleix els quatre requisits següents:

- que es pugui utilitzar amb qualsevol propòsit,
- que se'n puguin distribuir còpies,
- que se'n pugui analitzar el funcionament i modificar-lo per adaptar-lo a unes necessitats determinades (sense que deixi de ser lliure),
- que es pugui millorar i fer públiques les millores perquè d'altres se'n beneficiïn.

Aquests requisits no impliquen que el programari lliure hagi de ser gratuït (*freeware*), tot i que una bona part ho és. De fet, la seva distribució comercial és bàsica per poder finançar-ne el desenvolupament. El tercer requisit exigeix que el codi font del programari lliure sigui accessible per a tothom (*open source* o *codi obert*). D'altra banda, els documents creats amb gran part del programari lliure estan escrits en un format de tipus text sense codificar, de manera que, encara que el programari es deixés de mantenir o s'espatllés un document, la recuperació de la informació que conté seria una tasca relativament senzilla.

El 1985, Stallman i altres col·laboradors crearen la Free Software Foundation (FSF), un organisme no lucratiu dedicat a desenvolupar i difondre programari lliure, tot aprofitant el que s'havia desenvolupat fins aleshores. Per tal de preservar la "llibertat" del programari, van aprofitar la llei

de drets d'autor (*copyright*) donant-li la volta: van introduir en el programari lliure una clàusula que prohibeix als usuaris transformar-lo en programari propietari: el *copyleft*. El projecte principal de la FSF (en el qual Stallman treballava des de 1983) fou la creació d'un sistema operatiu i una col·lecció d'aplicacions bàsiques (editors, full de càlcul, compiladors, etc.) compatibles amb Unix, la qual cosa facilitaria la migració dels usuaris d'aquest sistema propietari. El nou programari, que es va anomenar *GNU* (GNU's Not Unix), havia de tenir, almenys, les prestacions de l'Unix, tot millorant algunes deficiències detectades en aquest sistema amb el pas dels anys. El 1990 el sistema GNU estava quasi complet; l'única part important que faltava era el nucli del sistema. Però el 1991, Linus Benedict Torvalds —que aleshores tenia 21 anys— desenvolupà un nucli compatible amb Unix per als processadors Intel dels PC que l'any següent es combinà amb el programari GNU per donar lloc a un sistema operatiu complet: el GNU/Linux.

Actualment, el GNU/Linux és un sistema força complet per al qual s'ha desenvolupat un programari d'aplicació molt divers i s'hi continua treballant de manera molt activa. En la bibliografia (§ D.3) s'indiquen diverses fonts que proporcionen el sistema operatiu juntament amb una àmplia selecció d'aplicacions.

C Xifres significatives, taules i gràfiques

Quan es fan tractaments de dades mitjançant programes informàtics és fàcil caure en la temptació d'expressar els resultats amb moltes xifres significatives sense tenir en compte la precisió de les dades de partida i/o la dels càlculs efectuats. És per això que hem cregut convenient recollir en aquest apèndix algunes pautes que convé seguir de cara a l'expressió de resultats, les quals són aplicables independentment del fet que aquests s'hagin obtingut o no amb l'ajut d'un ordinador. Atès que aquest recull serà força esquemàtic no discutirem la justificació d'aquestes pautes.

C.1 Xifres significatives

Anomenem xifres significatives d'una dada numèrica a les xifres (o dígits) que hi ha a partir de la primera xifra no nul·la. Per exemple, el valor 254,3 té quatre xifres significatives, però 254,32 o 254,30 en tenen cinc (els zeros a la dreta compten!). Els valors 0,000105 i 14,0 tenen tres xifres significatives, el mateix que 300 $\times 10^3$ i 3,00 $\times 10^5$ (aquestes dues notacions són equivalents); en canvi, 3×10^5 en té una sola.

El grau d'imprecisió o error absolut d'una magnitud física s'escriu amb dues xifres significatives com a màxim. Quan, en arrodonir l'error a dues xifres significatives, aquestes formen un nombre ≥ 16 es recomana expressar-lo amb una sola xifra; així, si l'error estimat és de $\pm 0,028$, és millor expressar-lo com a $\pm 0,03$. En canvi, si les dues xifres resultants formen un nombre comprès entre 10 i 15 expressarem aquell error amb dues xifres significatives; per exemple: $\pm 0,0013$.

Quan escrivim el valor d'una magnitud física hem de fer servir un nombre de xifres significatives tal que la darrera sigui del mateix ordre decimal que la darrera de les de l'error, tal com es veu en els exemples següents:

valors expressats incorrectament:	3,416 $\pm 0,14$	0,0085 $\pm 0,00010$	4 628 ± 374
valors expressats correctament:	3,42 $\pm 0,14$	0,00850 $\pm 0,00010$	$(46 \pm 4) \times 10^2$

L'error també es pot indicar entre parèntesis a continuació del valor de la magnitud: 3,42(14), 0,00850(10), 46(4) $\times 10^2$; en aquest cas, s'entén que l'error s'aplica a les últimes xifres del valor que el precedeix.

El valor d'una magnitud física s'expressa com un nombre seguit de la unitat a què correspon aquest nombre:

$$magnitud = nombre \times unitat \tag{C.1}$$

Per exemple:

$$P = 1013 \text{ hPa}; \quad \Delta H = 249,2 \text{ kJ mol}^{-1}$$

Quan es canvia la unitat s'ha de preservar el nombre de xifres significatives de la dada. Per exemple, l'entalpia de formació estàndard de l'oxigen atòmic a 25°C és de (249,18 $\pm 0,07$) kJ mol^{-1}; si volem expressar-la en J mol^{-1} no haurem d'escriure (249180 $\pm$ 70) J mol^{-1} sinó, per exemple, (249,18 $\pm 0,07) \times 10^3$ J mol^{-1}, (2,4918 $\pm 0,0007) \times 10^5$ J mol^{-1} o (24 918 $\pm 7) \times 10$ J mol^{-1}.

Quan escrivim el valor d'una magnitud física sense expressar-ne l'error s'entén que aquest és inferior a la unitat de la darrera xifra significativa. Així, la dada de l'exemple anterior es podria

expressar com a 249,2 kJ mol^{-1} o 249,2 $\times 10^3$ J mol^{-1}, però no com a 249,18 kJ mol^{-1}. Es pot precisar una mica més el valor sense indicar explícitament l'error donant una xifra significativa més com a subíndex, la qual cosa vol dir que l'error està comprès entre 1 i 9 unitats del dígit subindicat. Així, el valor anterior es pot expressar en la forma: $249{,}1_8$ kJ mol^{-1}.

És important tenir en compte que en operar amb dades sotmeses a error pot canviar el nombre de xifres significatives d'aquestes. El cas més típic es presenta quan restem dues dades properes entre elles. Per exemple, si calculem la durada d'un esdeveniment breu com a diferència entre els temps cronometrats del final i de l'inici de l'esdeveniment amb una precisió de 0,2 s —per exemple, $(222,7 \pm 0,2)$ s i $(216,3 \pm 0,2)$ s—:

$$\Delta t = (222,7 \pm 0,2)\ \text{s} - (216,3 \pm 0,2)\ \text{s} = (6,4 \pm 0,2)\ \text{s}$$

el resultat del càlcul té dues xifres significatives, la meitat que les dades. Tot i que els tres valors tenen el mateix error absolut (dues dècimes de segon), l'error relatiu serà més gran en el resultat que en les dades.

El nombre de xifres significatives d'una dada també pot canviar quan es canvia l'origen de l'escala. Per exemple, si hem mesurat una temperatura de 2,374°C amb un error de $\pm 0,005$°C i la convertim a temperatura absoluta expressada en kelvins, el nombre de xifres significatives passarà de 4 a 6, ja que l'error absolut no es veu afectat (s'ha de sumar el nombre exacte 273,15000...) i augmenta l'ordre de magnitud de la dada; per exemple:

$$t = (2,374 \pm 0,005)^\circ\text{C} \quad \Longrightarrow \quad T = [(2,374 \pm 0,005) + 273,15]\ \text{K} = (275,524 \pm 0,005)\ \text{K}$$

És fàcil comprovar que aquest augment de xifres significatives no es produeix quan la temperatura Celsius compleix $726{,}85^\circ\text{C} > t \geq 100^\circ\text{C}$ o $t \geq 1000^\circ$C; per exemple:

$$t = (726,82 \pm 0,10)^\circ\text{C} \quad \Longrightarrow \quad T = [(726,82 \pm 0,10) + 273,15]\ \text{K} = (999,97 \pm 0,10)\ \text{K}$$
$$t = (1000,000 \pm 0,002)^\circ\text{C} \quad \Longrightarrow \quad T = [(1000,000 \pm 0,002) + 273,15]\ \text{K} = (1273,152 \pm 0,002)\ \text{K}$$

A l'hora d'arrodonir la darrera xifra significativa escrita hi ha una ambigüitat quan totes les xifres que cal suprimir són 0 excepte la primera, que és un 5; llavors se sol agafar com a criteri que la darrera xifra que es conserva s'augmenta en 1 si és senar i no s'augmenta si és parella. Així, en arrodonir el valor (6,7350 ± 0,12) kg s'haurà de conservar la segona xifra decimal; com que aquesta (un 3) va seguida d'un 5 i és senar s'augmentarà en 1, i el valor arrodonit s'expressarà de la forma (6,74 ± 0,12) kg. En canvi, en arrodonir el valor (6,7450 ± 0,12) kg, la darrera xifra que s'ha de conservar és parella (el 4), i no es modificarà. El valor arrodonit serà, doncs, el mateix que abans: (6,74 ± 0,12) kg.

És important tenir en compte que mai no s'han d'arrodonir els resultats intermedis d'un càlcul. Per tal d'introduir el mínim error possible en el tractament numèric de les dades, s'ha de procurar fer els càlculs amb la màxima precisió disponible (convé que aquesta sigui superior a la de les dades), i fer els arrodoniments que calguin només a l'hora de presentar els resultats finals o publicar-los en algun document. Aquesta precaució afecta principalment els càlculs fets a mà, ja que en un programa informàtic els resultats intermedis es guarden en variables reals (si tenen decimals) amb la precisió que aquestes admeten. No obstant això, si es vol treballar amb precisió doble per tal de minimitzar l'error de càlcul convé que totes les variables i constants que intervenen en el càlcul tinguin aquella precisió (vegeu l'exemple de la pàgina 30).

C.2 Criteris per construir taules

Per no haver d'expressar les unitats en cada dada d'una taula, aquestes s'expressen com a nombres adimensionals que representen valors de la magnitud corresponent dividits per la unitat utilitzada per expressar-los (vegeu l'equació (C.1)):

$$\textit{nombre} = \textit{magnitud}/\textit{unitat}$$

L'encapçalament de cada columna (o fila) ha d'indicar el que representen aquests nombres; per exemple: si representen densitats expressades en g/cm^3 l'encapçalament haurà de ser ρ / g cm^{-3}.

S'han d'evitar les expressions amb més d'un signe de divisió, com ara ρ / g / cm^3. Si els valors s'expressen dividits per un factor constant, és recomanable incloure aquest valor juntament amb les unitats; així, les dues columnes de la taula següent expressen el mateix:

	ρ / g cm^{-3}	ρ / 10^3 kg m^{-3}
alumini	2,698	2,698
coure	8,960	8,960
or	19,320	19,320
ferro	7,874	7,874
plata	10,500	10,500

Per exemple, la densitat de la plata és:

$$\rho = 10,500 \text{ g cm}^{-3} = 10,500 \times 10^3 \text{ kg m}^{-3}$$

C.3 Criteris per construir gràfiques

A les gràfiques es representaran nombres adimensionals que representen, com en el cas de les taules, magnituds físiques dividides per la unitat corresponent. A cada eix d'una gràfica s'ha d'indicar la magnitud representada i les seves unitats, de la mateixa manera que en elsencapçalaments de les taules. L'escala dels eixos s'agafarà de manera que la longitud de cada eix coincideixi amb (o bé sigui una mica més gran que) l'interval de valors de la variable que s'hi representa. D'aquesta manera, si l'eix horitzontal representa valors experimentals d'una magnitud x i el vertical representa els d'una altra magnitud y aproximadament proporcional a x, la línia recta que s'ajusta millor als punts de la gràfica (la recta de regressió), tindrà una inclinació d'uns 45°.

La representació gràfica de parelles de valors experimentals es farà d'acord amb els criteris següents:

- El valor central estimat per a cada punt (x_i, y_i) s'ha d'indicar amb un petit cercle o un símbol que determini un punt (per exemple, $\times$) suficientment gran perquè es pugui veure, però no massa (evitar la teoria del "punt gros", segons la qual totes les representacions són lineals si dibuixem els punts suficientment grossos).
- Cada punt s'ha d'acompanyar amb el corresponent rectangle d'error $\{x_i \pm s(x_i),\ y_i \pm s(y_i)\}$ però, com que normalment suposarem que només la variable y està sotmesa a error, aquest rectangle d'error quedarà reduït a un segment vertical centrat en el punt i de longitud $2s(y_i)$ que se superposarà al símbol del punt.
- Si els errors $s(y_i)$ són inapreciables, es deixarà només el símbol amb què es representen els punts.

D Bibliografia

Aquest apèndix recull una relació de textos i pàgines *web* que permeten ampliar en diferents aspectes la informació continguda en aquest llibre.

D.1 Enciclopèdies i diccionaris generals

D.1.1 Català

- *http://ca.wikipedia.org/*
- *http://ca.wiktionary.org/*
- *http://dcvb.iecat.net/*
- *http://www.termcat.net/cercaterm/*

D.1.2 Castellà

- *http://es.wikipedia.org/*
- *http://es.wiktionary.org/*

D.1.3 Anglès

- *http://en.wikipedia.org/*
- *http://en.wiktionary.org/*
- *http://www.wordiq.com/*

D.1.4 Recopilació de diccionaris

- *http://www.onelook.com*

D.1.5 Diccionari anglès-català

- *http://dacco.sourceforge.net/*

D.2 Diccionaris de termes informàtics

D.2.1 Català

- *http://www.softcatala.org/projectes/dicci/*
- *http://www.softcatala.org/projectes/eines/recull/3.0/recull.htm*
- *http://www.xtec.es/~jrosell3/ticcionari/*

D.2.2 Castellà

- *http://www.tugurium.com/gti/*

D.2.3 Anglès

- *http://www.webopedia.com/*
- *http://wombat.doc.ic.ac.uk/foldoc/*

D.3 Distribucions de GNU/Linux

- *Debian*: *http://www.debian.org, http://www.debian.org/index.ca.html* o *http://www.debian.org/index.es.html*
- *Ubuntu*: *http://www.ubuntu.com*
- *RedHat*: *http://www.redhat.com* o
- *SuSe*: *http://www.novell.com/linux* o
- *Mandriva*: *http://www.mandriva.com* o
- *Knoppix*: *http://www.knoppix.org* o *http://www.knoppix-es.org*

D.4 Fortran 77

- Especificacions del Fortran 77 estàndard: *American National Standard Institute, Publication X3.9*, 1978 (*http://www.fortran.com/fortran/F77_std/rjcnf.html*).
- Manual del compilador g77 (versió GCC-3.4.6): *http://gcc.gnu.org/onlinedocs/gcc-3.4.6/g77/*
- Manual del compilador gfortran: *http://gcc.gnu.org/onlinedocs/gfortran/*
- Lignelet, P.: *Fortran 77. Lenguaje Fortran V*, Masson, Barcelona, 1985.
- Lignelet, P.: *La pratique du Fortran 77*, Masson, Barcelona, 1982.

D.5 Aplicacions

- *Basic Linear Algebra Subprograms* (BLAS): *http://www.netlib.org/blas/*
- DeVries, P. L.: *A First Course in Computational Physics*, Wiley, New York, 1994.
- Johnson, K. F.: *Numerical Methods in Chemistry*, Marcel Dekker, New York, 1980.
- Norris, A. C.: *Computational Chemistry. An Introduction to Numerical Methods*, Wiley, Chichester, 1986.
- *PGplot Graphics Subroutine Library*: *http://www.astro.caltech.edu/~tjp/pgplot/*

- Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P.: *Numerical Recipes in Fortran. The Art of Scientific Computing*, 2nd ed., Cambridge University Press, Cambridge, 1992 (versió en línia: *http://www.nrbook.com/a/bookfpdf.php*).

- *The NAG Fortran Library, Mark 21.1*, The Numerical Algorithms Group Limited, Wilkinson House, Jordan Hill Road, Oxford, U. K., OX2 8DR, 2006 (*http://www.nag.co.uk/*).

TEXTOS DOCENTS

328

Aquest llibre pretén aproximar el lector a la informàtica i introduir-lo en un llenguatge de programació d'àmbit cientificotècnic. El text està orientat a no professionals de la informàtica —estudiants, graduats i llicenciats en estudis científics o tècnics (física, química, biologia, geologia, enginyeries, etc.)— que vulguin tenir unes nocions de programació suficients per poder elaborar programes senzills o de dificultat mitjana. L'entorn de treball serà el que proporcionen les distribucions del sistema operatiu GNU/Linux, el qual pot estar instal·lat al disc dur o bé carregar-se des d'un CD autònom (live CD). El llenguatge escollit és el Fortran 77, que és senzill i clar, i està molt estès en el camp de les ciències fisicoquímiques.

Els conceptes es presenten de manera esglaonada: d'entrada, s'introdueixen amb l'ajut d'exemples fàcils i, a mesura que s'avança en l'explicació, es tracten aplicacions més complexes. La major part dels exercicis proposats inclouen la solució o bé un conjunt de dades i resultats que permetin verificar el funcionament correcte del programa. L'objectiu és que el profà en la matèria pugui iniciar-se de seguida en la pràctica de la programació i que aconsegueixi un nivell que li permeti abordar sense dificultats la lectura de manuals més complets.

www.publicacions.ub.edu

Publicacions i Edicions

UNIVERSITAT DE BARCELONA