Fitxers. Accés seqüencial.

5

Isidre Guixà i Miranda
IES SEP Milà i Fontanals, d'Igualada

Programació estructurada i modular

Febrer del 2008
© Isidre Guixà i Miranda
IES SEP Milà i Fontanals
C/. Emili Vallès, 4
08700 - Igualada

En cas de suggeriment i/o detecció d'error podeu posar-vos en contacte via el correu electrònic iguixa@xtec.cat

Índex

Introducció

Les aplicacions informàtiques, no importa el sector al que pertanyin, tenen com a punt en comú, la gestió de dades i acostuma a interessar el poder-les mantenir en suport de memòria externa, ja sigui mentre l'aplicació està en execució, ja sigui quan l'aplicació està aturada.

Per aquest motiu apareix la necessitat de poder enregistrar les dades en suport de memòria externa i la possibilitat que faciliten els sistemes operatius no és altra que enregistrar les dades en fitxers dins el sistema de fitxers que gestiona el sistema operatiu.

Així doncs, en una primera fase de la història de la programació informàtica, els programadors desenvolupaven aplicacions que enregistraven les dades en suport de memòria externa i els programes s'encarregaven de controlar els suports de memòria externa. Això provocava que les aplicacions informàtiques depenguessin del tipus de suport extern i quan aquest canviava, calia modificar els programes. Evidentment, això era un greu problema per al manteniment de les aplicacions informàtiques.

L'evolució en el desenvolupament de les aplicacions informàtiques va passar per l'aparició de conjunts de programes que s'encarregaven del control del suport extern i que permetien als programadors despreocupar-se'n en els seus programes. Aquests conjunts de programes es coneixen com els sistemes gestors de fitxers i en aquesta unitat didàctica pretenem introduir-nos-hi.

En el nucli d'activitat "La informació en memòria externa: els fitxers" presentem un seguit de conceptes bàsics vinculats a la gestió de fitxers. No pretén ser una bíblia sobre el tema. Heu de considerar aquest nucli d'activitat com els conceptes bàsics a conèixer per a poder iniciar el desenvolupament de programes que gestionin fitxers. I, en particular, classificarem els sistemes gestors de fitxers, segons el tipus d'accés a les dades que permeten, en fitxers seqüencials, fitxers relatius i fitxers seqüencial-indexats.

En el nucli d'activitat "Gestió de fitxers via el llenguatge C" mostrem les possibilitats que facilita el llenguatge C per a gestionar els fitxers existents en el sistema de fitxers del sistema operatiu. Hi ha llenguatges de programació, com per exemple *Cobol*, que aporten un sistema gestor de fitxers (seqüencial i/o relatiu i/o seqüencial indexat). No és el cas del llenguatge C. El llenguatge C simplement ens facilita un conjunt de

funcions per poder gestionar els fitxers del sistema de fitxers del sistema operatiu i haurà de ser el programador qui dissenyi els mecanismes adequats per accedir a les dades (simulant fitxers seqüencials i/o relatius i/o seqüencial-indexats).

Per últim, en el nucli d'activitat "Gestió de fitxers seqüencials" introduïm les operacions bàsiques que faciliten els sistemes gestors de fitxers seqüencials i els algorismes bàsics de tractament seqüencial que ha de dominar tot programador, els quals durem a la pràctica amb la utilització del llenguatge C.

Per aconseguir els nostres objectius, heu de reproduir en el vostre ordinador i, a ser possible, en diverses plataformes, tots els exemples incorporats en el text, per a la qual cosa, en la secció "Recursos de contingut", trobareu tots els fitxers necessaris, a més de les activitats i els exercicis d'autoavaluació.

Objectius

A l'acabament d'aquesta unitat didàctica, l'estudiant ha de ser capaç de:

1. Diferenciar la utilitat de la memòria externa i la de la memòria interna.
2. Identificar els objectius a assolir amb la utilització de sistemes gestors de fitxers.
3. Conèixer els elements i les característiques dels sistemes gestors de fitxers.
4. Identificar els diferents tipus d'accés sobre fitxers.
5. Distingir els mecanismes que els sistemes gestors de fitxers aporten per a garantir la independència física de les dades.
6. Distingir el tipus d'accés dels fitxers seqüencials.
7. Identificar les operacions que faciliten els sistemes gestors de fitxers seqüencials.
8. Dissenyar algorismes sobre fitxers seqüencials per a donar resposta a les múltiples necessitats de tractament d'informació en el món real.
9. Desenvolupar programes que gestionin fitxers seqüencials, efectuant altes, baixes, consultes i modificacions.
10. Utilitzar, de manera correcta i eficient, la gestió seqüencial de fitxers facilita el llenguatge C.

1. La informació en memòria externa: els fitxers.

Ens proposem introduir els conceptes teòrics bàsics que cal conèixer per poder-nos iniciar en la gestió de fitxers per a enregistrar les dades que cal mantenir entre diferents execucions dels programes. Així, ens cal:

- Tenir clara la distinció entre memòria interna i memòria externa
- Identificar els conceptes vinculats a la representació de la informació (entitat, atribut, valor, domini, identificador, clau) i els corresponents conceptes en el camp dels sistemes gestors de bases de dades.
- Conèixer el funcionament que s'estableix entre els programes i els sistemes gestors de fitxers.
- Distingir els diferents tipus de sistemes gestors de fitxers que podrien existir segons les combinacions dels tipus d'accés a les dades que permeten i identificar els que es poden trobar en la realitat.

1.1. Memòria interna i memòria externa

Els programes informàtics gestionen dades. En alguns casos, en finalitzar l'execució del programa, no hi ha cap necessitat de guardar una còpia de les dades gestionades (d'un subconjunt o de la totalitat), però hi ha nombroses aplicacions en què les dades tractades han de ser conservades fins i tot després de la fi de l'execució. Com podria una empresa examinar la seva gestió si les dades relatives a les vendes no existissin després de generar les factures en paper? Sorgeix, per tant, la necessitat de conservar les dades fora de la memòria interna de l'ordinador, en un suport permanent, és a dir, en memòria externa.

D'altra banda, el volum de certes dades és tan gros que no és possible mantenir-les en memòria interna. Continuant en l'entorn empresarial, si es vol fer algun tipus de gestió a partir de les factures d'un client, no té cap sentit haver de tenir tots els clients en memòria interna, cosa probablement impossible a causa de l'espai de memòria que requeriria, quan en realitat només es necessiten les dades referents a un client.

Així doncs, l'existència de la memòria externa queda justificada per les raons següents: ‼

- L'emmagatzematge permanent de les dades, és a dir, un suport no volàtil.

- L'agrupació d'un gros volum de dades que puguin ser separades per a la seva gestió, és a dir, la capacitat il·limitada.

- Un cost menys elevat que la memòria interna.

Els tipus de memòries actuals són dispositius d'emmagatzematge magnètic o d'emmagatzematge òptic.

Argot informàtic

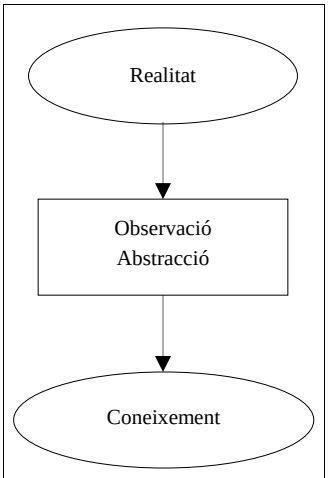
Els termes anglesos comunament utilitzats pels informàtics per a referir-se als dispositius de memòria externa són: streamer o tape per a la cinta magnètica; floppy disk per als discos flexibles; hard disk per als discos durs, i optical disk per als discos òptics.

Els dispositius magnètics actuals de memòria externa són les cintes magnètiques (cada cop menys usades), els discos flexibles i els discos durs. També hi ha els discos òptics (cada cop més utilitzats).

Podríem dedicar molt espai a presentar els diferents tipus de dispositius, però no és la finalitat d'aquest crèdit tot i que sí forma part dels coneixements que tot informàtic ha de tenir. Podeu trobar molta informació sobre aquest tema i, sobretot, si voleu estar al dia, heu d'anar llegint la premsa informàtica. La tecnologia de suports és un dels camps en contínua evolució a causa de la ingent quantitat de memòria necessària per a emmagatzemar la informació que tothom vol tenir a l'abast. Oi que heu sentit més d'una vegada que estem davant l'era de la informació? (!!)

1.2. Representació de la informació

L'ésser humà, mitjançant els sentits, observa la realitat, de la qual extreu, a través d'un procés d'abstracció, el coneixement, el qual es pot considerar transmissible o no transmissible. (!!)



Representació de la informació

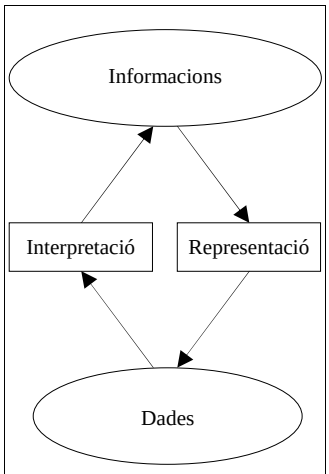
Ja hem dit moltes vegades que la informàtica persegueix el tractament de grans volums d'informació amb agilitat i eficiència.

Anomenem **informació** qualsevol coneixement transmissible.

Per tant, la informació és un concepte abstracte al qual hem de donar una representació si volem fer-ne un tractament informàtic.

Anomenem **dada** la representació d'una informació.

Aquesta definició no es contradiu amb la que s'acostuma a utilitzar en la iniciació en el mon de la programació informàtica on s'acostuma a dir que una "dada és qualsevol informació que utilitza l'ordinador".
És a dir, d'una banda tenim les informacions (món dels coneixements) i, de l'altra, tenim les dades (món de les representacions). Una informació es representa en una dada i a partir d'una dada s'interpreta una informació.



Relació entre informacions i dades

Es clar que:

- Persones diferents poden arribar a informacions diferents a partir d'una mateixa realitat. Això és degut al frame (marc de referència).

- La representació d'una informació en una dada no és única.

A partir d'això deduïm l'existència de tres mons:

- Món real, objectiu, únic per a tothom.
- Món dels coneixements, subjectiu, plural.
- Món de les representacions, comunicatiu.

Tots tenim clar que en el món dels coneixements s'hi inclouen dos tipus d'informacions:

- Elemental, formada per subjecte + verb + predicat.

- Complexa, formada per la composició de dues o més informacions elementals a partir de les conjuncions.

Però, i en el món de les representacions? En la iniciació a la programació informàtica ens trobàvem exemples en els quals preníem decisions sobre els tipus de dades que havíem d'utilitzar per a representar les informacions que calia gestionar. En alguns moments apareixien exemples en els quals havíem de tenir en compte elements repetits. I no varem tenir problemes per sortir-nos-en. Ens manca, però, utilitzar de manera adequada els conceptes informàtics que es fan servir en la representació de la informació en dades.

1.3. Entitat, atribut, valor, domini, identificador, clau

Situem-nos en el món de les informacions. Hem dit que n'hi ha d'elementals (subjecte + verb + predicat) i de compostes. Aprofundim-hi una mica i millorem aquesta definició.

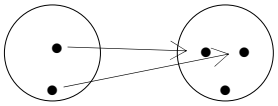
Una entitat és quelcom real, identificable i distingible.

Així, tota persona és una entitat, com també ho és un gos, una casa, un cotxe... Podem agrupar les entitats d'un mateix tipus sota un nom comú i llavors parlem de l'entitat persones, l'entitat gossos, l'entitat cases, l'entitat cotxes. Quan parlem d'un element concret d'una entitat acostumem a utilitzar el concepte d'instància.

Recordatori matemàtic

Una aplicació és una correspondència entre dos conjunts de manera que cada element del conjunt origen té una imatge en el conjunt destí.

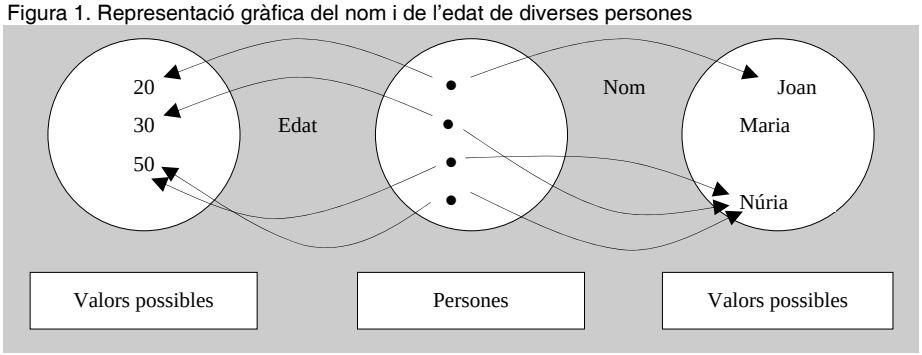
Exemple:



Un atribut és una aplicació d'una entitat (conjunt d'elements del mateix tipus) sobre un conjunt de valors.

Observeu l'analogia que hi ha entre "subjecte + verb + predicat" i "entitat + atribut + valor"?

Figura 1. Representació gràfica del nom i de l'edat de diverses persones



Observeu, a la figura 1, l'entitat persones, que conté quatre instàncies (els punts negres). Cada persona és "quelcom" real, distingible i identificable. Observeu els dos atributs: edat i nom. En podríem considerar d'altres: DNI, adreça, pes...

L'atribut edat és una aplicació del conjunt de les persones sobre el conjunt de valors possibles com a edat. Convindreu que el conjunt de valors ha de ser, en aquest cas, un conjunt de nombres naturals.

L'atribut nom és una aplicació del conjunt de les persones sobre el conjunt de valors possibles com a nom, és a dir, el conjunt de noms de persones.

Observeu que hi pot haver noms que no siguin la imatge de cap persona (Maria) i altres que poden ser la imatge de diverses persones (Núria). El mateix pot succeir amb l'atribut edat, tot i que en aquesta situació concreta no es produeix.

Anomenem domini d'un atribut el conjunt de valors possibles que pot prendre l'atribut, és a dir, que poden ser imatge d'alguna entitat per a l'atribut considerat.

Així, en l'exemple anterior tenim clar que l'atribut edat té com a domini el conjunt dels nombres naturals, ja que podem convenir que l'edat de les persones ha de ser numèrica, no ha de ser negativa i no ha de tenir decimals (això ja és més discutible...). Podem afinar més i considerar que l'edat de les persones pot tenir un límit superior, que podríem fixar en 200. Us sembla correcte?

D'altra banda, l'atribut nom té com a domini el conjunt de noms possibles en les persones. Evidentment, és el conjunt de cadenes constituït per les cadenes que tenen sentit com a nom de persones.

Hi ha diferents situacions que causen l'existència d'un valor nul: (!!)

- Quan el valor de l'atribut d'una entitat és desconegut, com per exemple l'edat d'una persona.

- Quan l'atribut no és aplicable sobre una instància concreta d'una entitat, com per exemple quants parts ha tingut una persona quan la persona és de sexe masculí.

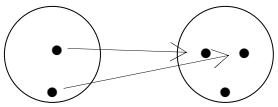
D'aquesta manera aconseguim mantenir el concepte d'atribut ja que com que ha de ser una aplicació, tota instància ha de tenir un valor com a imatge per a l'atribut. En les situacions anteriors es considera el valor nul.

Una aplicació injectiva és una aplicació en què cada element del conjunt destí és imatge d'un element del conjunt origen o bé no ho és de cap.

Recordatori matemàtic

Una aplicació injectiva és una aplicació en què cada element del conjunt destí és imatge d'un element del conjunt origen o bé no ho és de cap.

Exemple:



Un atribut és identificador quan l'aplicació és injectiva.

El concepte d'atribut identificador es correspon amb el fet que l'atribut no pugui tenir valors repetits. És un concepte semàntic, és a dir, no s'és atribut identificador en funció del cas particular que s'estigui tractant sinó del significat que l'atribut porta en si mateix. (!!)

En el cas particular del darrer exemple, l'atribut edat es comportava com a identificador, ja que no hi havia dues persones que tinguessin la

mateixa edat. Ara bé, sabem que aquesta situació no és la que porta vinculada l'atribut edat, ja que de tots és conegut que hi ha moltes persones que tenen la mateixa edat. Per tant, l'atribut edat no és identificador. Seguint amb el mateix exemple, en la situació particular presentada s'observa que l'atribut nom no és identificador ja que Núria és el nom de tres persones.

És molt important poder tenir, en tota entitat, com a mínim, un atribut identificador, ja que d'aquesta manera tenim l'entitat totalment identificada, valgui la redundància.

En el cas de l'entitat persones, quin identificador podem considerar? Cal tenir en compte l'entorn en el qual s'està parlant. Hi ha el costum d'utilitzar el DNI com a identificador de les persones. És correcte? Depèn! Si estem parlant de persones a escala mundial, no és correcte, ja que el DNI és un concepte de l'estat espanyol. D'altra banda, si pretenem gestionar persones en les quals hi pot haver infants, no és correcte, ja que en l'actualitat els infants no estan obligats a tenir adjudicat un DNI. I, per acabar, si gestionem persones per encàrrec de qualsevol de les administracions de l'estat espanyol, el DNI tampoc no és correcte com a identificador, ja que hi ha hagut errors en l'adjudicació i hi ha valors repetits. En resum, cal fer un bon estudi sobre un atribut per a poder-lo prendre com a identificador.

DNI

Sigles corresponents al document nacional d'identitat que l'estat espanyol adjudica als seus ciutadans. Tots els països tenen algun concepte identificador equivalent.

Fins fa poc, aquest document era obligatori a partir de l'edat de catorze anys. En l'actualitat, hi ha la intenció que sigui lliurat en el moment que es produeix el naixement.

Degut a errors administratius, a l'estat espanyol el DNI no és identificador, és a dir, hi ha diferents persones amb el mateix número de DNI. Aquesta situació, però, va desapareixent.

I si no disposem de cap atribut que per si sol sigui identificador? Intentarem, en aquest cas, utilitzar el concepte més genèric de clau.

> Una clau és un o diversos atributs que conjuntament identifiquen, de manera unívoca, una entitat.

Com a exemple, suposem que volem identificar les ciutats amb el seu nom. De tots és conegut que això no és possible ja que hi pot haver ciutats amb el mateix nom en diferents països. Fins i tot en un mateix país. Oblidem-nos d'aquesta darrera possibilitat. Suposem que en un país, el nom de la ciutat és un atribut identificador. Però hem de gestionar ciutats de diferents països. Quin identificador hem de prendre? Podem considerar la clau formada per la parella nom de país i nom de ciutat, parella que identifica tota ciutat.

Fixeu-vos que el concepte de clau també engloba el concepte d'atribut identificador, ja que una clau pot ser formada per un únic atribut.

Cal distingir els tipus de claus següents: (!!)

- Clau candidata: qualsevol de les claus possibles sobre una entitat.

- Clau mínima: clau que no té atributs redundants, és a dir, que tot atribut és imprescindible perquè sigui una clau.

- Clau primària: clau escollida per a identificar l'entitat. És única.

- Clau alternativa: clau candidata no primària.

No cal dir que cap atribut d'una clau no pot tenir definit el valor nul. (!!)

Donada una entitat, quina nomenclatura hem d'utilitzar per a indicar els atributs definits sobre ella? Utilitzarem la següent, molt corrent en el món informàtic:

entitat (atribut_1, atribut_2, atribut_3, ..., atribut_n)

Considerarem que el noms entitat, atribut_1, atribut_2, ... atribut_n no poden contenir cap espai. Indicarem la clau primària subratllant el conjunt d'atributs que la formin. Els atributs que permetin l'existència de valor nul aniran acompanyats del subíndex VN a la dreta. Així, en l'exemple de les ciutats del món posaríem:

ciutats (nomPaís, nomCiutat, superfície$_{VN}$, nHabitants$_{VN}$, ...)

Tots els conceptes exposats són genèrics al món de les representacions. Quan hàgim de fer un programa utilitzant dades estàtiques per a gestionar ciutats del món, probablement escollirem la definició de dades següent:

```
const MAX_CIU = 100; ficonst
tipus ciutat = tupla
                 nomPaís, nomCiutat : cadena [30];
                 superfície : real;  /* Valor Nul = 0 */
                 nHabitants : enter; /* Valor Nul = -1 */
               fitupla
fitipus
var tciu = taula [MAX_CIU] de ciutat; fivar
```

Observem que els atributs `superfície` i `nHabitants` permeten l'existència de valor nul, ja que en la nostra gestió de ciutats del món volem gaudir de la possibilitat de tenir ciutats de les quals encara no coneguem la superfície o el nombre d'habitants. No n'hi ha prou dient que aquest atribut permet valor nul. Aquest és un concepte abstracte que, en traslladar-lo al món de les representacions, es converteix en una dada que té el significat de valor nul. Per això, sempre que es parla de valor nul per a un atribut, cal definir quina dada haurà de representar

aquest valor. En l'exemple anterior hem definit que el valor 0 seria la representació del valor nul en l'atribut superfície (cosa perfectament possible ja que no hi pot haver cap ciutat amb superfície zero). En canvi, hem definit el valor -1 com a representació del valor nul per a l'atribut nHabitants (ja que hem considerat que podríem tenir una ciutat totalment deshabitada, és a dir, amb el valor 0 pertanyent al domini) i això ens ha obligat a definir l'atribut del tipus enter.

1.4. Camp, registre, fitxer, base de dades

Implementació

Implementació és un terme molt utilitzat en informàtica per a indicar la posada en pràctica d'un concepte abstracte. Es diu, per exemple, que s'està implementant una solució a un problema quan es passa del món abstracte –en el qual s'ha dissenyat la solució– al món pràctic –on podrem executar la solució.

La definició següent ens dona una implementació del disseny de l'entitat ciutats sobre dades estàtiques:

```
const MAX_CIU = 100; ficonst
tipus ciutat = tupla
                 nomPaís, nomCiutat : cadena [30];
                 superfície : real;  /* Valor Nul = 0 */
                 nHabitants : enter; /* Valor Nul = -1 */
               fitupla
fitipus
var tciu = taula [MAX_CIU] de ciutat; fivar
```

Centrem-nos, però, en el nostre objectiu: la gestió de fitxers. Les implementacions dels dissenys d'entitats sobre fitxers utilitzen nous conceptes que definirem a continuació.

Un camp és la representació del valor que pot prendre un atribut sobre una entitat.

Un registre és la representació d'una instància d'entitat.

Un fitxer és la representació d'una entitat (conjunt d'instàncies).

Utilitzant l'exemple de les ciutats del món, tindríem que:

- `nomPaís`, `nomCiutat`, `superfície`, `nHabitants` són camps.
- El conjunt de valors corresponents a cada ciutat emmagatzemats en els camps formen un registre.
- El conjunt de registres corresponents a les diferents ciutats del món formen el fitxer.

Observeu, a la taula 1, el conjunt de ciutats del món que s'ha d'emmagatzemar en un fitxer.

Taula 1. Conjunt de ciutats del món a emmagatzemar en un fitxer

nomPaís	nomCiutat	superfície	nHabitants
Espanya	Madrid	0	-1
Veneçuela	Barcelona	0	-1
Catalunya	Girona	0	-1
Catalunya	Lleida	0	-1
Catalunya	Tarragona	0	-1
Catalunya	Barcelona	0	-1

Cada columna es correspon amb un camp, ja que conté els valors que pren cadascun dels atributs definits sobre l'entitat. Cada fila es correspon amb un registre, ja que conté el conjunt de valors que prenen els diferents atributs sobre una ciutat en concret (una instància de l'entitat ciutats del món). El conjunt de files es correspon amb el fitxer, el qual emmagatzema el conjunt d'instàncies de l'entitat ciutats del món.

Observeu l'analogia que hi ha entre la implementació efectuada sobre dades estàtiques al final de l'apartat anterior i la implementació presentada ara en fitxers? La taula 2 ens la mostra.

Taula 2. Analogia entre dades implementades en fitxers i en dades estàtiques.

Fitxer	Dades estàtiques
Camp	Apartat de tupla
Registre	Fila de taula (tupla)
Fitxer	Taula

Per acabar, introduirem el concepte de base de dades.

Una base de dades és un conjunt de fitxers relacionats entre ells que disposa d'unes eines (sistema gestor de bases de dades) per a poder-los gestionar de manera eficient.

Els sistemes gestors de bases de dades són fruit de l'evolució lògica dels sistemes gestors de fitxers, el nostre objectiu actual. Quan estudieu els sistemes gestors de bases de dades podreu observar la diferència entre els dos tipus de sistemes gestors.

Termes anglesos

El terme anglès utilitzat per a fer referència a un sistema gestor de fitxers és File Management System, que s'acostuma a abreujar FMS. En canvi, el terme utilitzat per a fer referència a un sistema gestor de bases de dades és Database Management System, que s'abreuja DBMS.

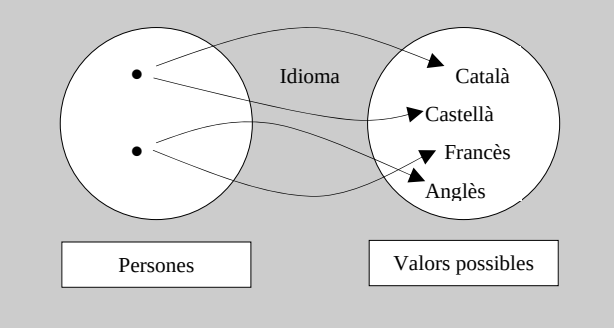
En les aplicacions que treballen amb sistemes gestors de fitxers és molt lògic utilitzar el terme base de dades per a fer referència al conjunt de fitxers on s'emmagatzemen les dades que gestiona l'aplicació. No heu de confondre la utilització d'aquest terme en aquest context amb una

veritable base de dades, concepte que fa referència a les dades emmagatzemades per mitjà d'un sistema gestor de base de dades.

1.5. Atributs multivalor

Se'ns presenta un problema. Suposem que volem representar l'entitat persona i volem gestionar els idiomes que la persona coneix. La figura 2 ens representa la situació que es pot presentar.

Figura 2. Representació dels idiomes que coneixen les persones



Tal com es veu en la figura 2, ens podem trobar persones que coneguin diversos idiomes. Com podem tractar-ho? La correspondència presentada en el dibuix no és una aplicació, ja que algun element (en aquest cas les dues persones) del conjunt origen té més d'una imatge en el conjunt destí. Per tant, idioma no és un atribut! Com se soluciona aquesta situació? Tenim dues possibilitats.

a) Tractar el concepte d'idioma com un "atribut multivalor" (no és aplicació!) considerant que la seva representació no sigui un únic valor, sinó un conjunt de valors fins a un nombre màxim i encabir els possibles valors en una taula. En aquest cas, el tipus de dada del camp corresponent seria una taula.

Taula 3. Implementació de persona. El camp idioma és una taula de tres posicions.

DNI	Nom	Idioma		
11111111	Teresa	Català	Castellà	Anglès
22222222	Oriol	Català	Castellà	
33333333	Guifré	Català		

La taula 3 ens mostra la implementació de persona que contempla el camp idioma com una taula de tres posicions, on cada posició conté un idioma. Ja us deveu imaginar els problemes que això porta:

- Què passa quan una persona coneix més de tres idiomes?
- Es perd espai quan la persona coneix menys de tres idiomes.

La solució presentada és vàlida per a representar atributs multivalor quan el nombre de valors és fix, com succeeix si es vol gestionar el volum mensual de vendes de cada producte.

Taula 4. Exemple d'atribut representat per una taula que utilitza totes les seves posicions.

Codi	**Descripció**	**Vendes mensuals**											
PAT	Patata	300	200	500	800	625	433	215	923	427	825	999	0
PER	Pera	415	234	432	892	234	982	298	126	836	183	276	456
POM	Poma	25	87	12	98	23	65	12	99	26	78	21	54

La taula 4 ens mostra un exemple en el que l'atribut "vendes mensuals" està representat en una taula de dotze posicions (una per cada mes). En aquesta situació, la implementació no té els problemes que hem trobat en l'exemple dels idiomes.

En qualsevol cas, cal rebutjar aquest tipus de solució, tot i que us trobareu amb aplicacions que la tenen implementada.

Si es té un nombre fix de valors que cal mantenir per a qualsevol registre, la solució és considerar tants atributs com valors possibles, com mostra la taula 5.

Taula 5. Exemple d'atribut que pot tenir varis valors representat per diferents camps.

Codi	Descripció	v01	v02	v03	v04	v05	v06	v07	v08	v09	v10	v11	v12
PAT	Patata	300	200	500	800	625	433	215	923	427	825	999	0
PER	Pera	415	234	432	892	234	982	298	126	836	183	276	456
POM	Poma	25	87	12	98	23	65	12	99	26	78	21	54

Observeu que d'aquesta manera tots els atributs són vertaders atributs (aplicacions).

Si el nombre de valors és variable, cal adoptar la solució de l'apartat següent.

b) Gestionar l'atribut problemàtic amb l'ajut d'un nou fitxer que contingui, per a cada registre amb diversos valors per a l'atribut, tants registres com valors existeixin. Fixeu-vos en l'exemple de la taula 6.

És clar que per a poder fer aquest tractament és necessari tenir definida una clau primària en el fitxer inicial, ja que és la que ens permet lligar cada instància de l'entitat persona amb els diferents idiomes. En aquest cas hem considerat el DNI com a clau primària de persones.

Taula 6 . Solució per a la implementació dels idiomes que coneixen les persones.

Fitxer de persones	
DNI	**Nom**
11111111	Teresa
22222222	Oriol
33333333	Guifré

Fitxer d'idiomes coneguts per les persones	
DNI	**Idioma**
11111111	Català
11111111	Castellà
11111111	Anglès
22222222	Català
22222222	Castellà
33333333	Català

1.6. Classificació funcional dels fitxers

Segons la funció que tinguin, els fitxers es poden classificar en tres grups:

1) Permanents

Són fitxers de llarga durada que contenen informació que varia molt poc en el temps. En alguns casos és necessari actualitzar-los periòdicament. Se subdivideixen en tres grups:

a) De constants

La seva informació es manté pràcticament inamovible. S'utilitzen principalment com a consulta. Poden considerar-se'n dos subtipus:

- Els que no s'actualitzen mai, com les províncies d'un país i la taula de logaritmes.

- Els que s'actualitzen molt poc, com les taules de l'impost de la renda de les persones físiques i les taules de cotització a la Seguretat Social, que s'actualitzen una vegada l'any.

b) Mestres

Són els fitxers vitals per a l'aplicació informàtica i contenen la informació actualitzada, com en el cas dels clients, els venedors, els articles, els proveïdors...

Els registres d'aquests fitxers s'han d'anar actualitzant (altes, baixes i modificacions), cosa que es pot fer de dues maneres:

- interactiva, en temps real (processos on line o en línia),
- amb fitxer de moviments, en temps diferit (processos batch).

Actualitzacions en temps diferit, en el moment actual

La revolució de les comunicacions ha permès oblidar l'actualització dels fitxers mestres en temps diferit.

En la dècada dels setanta i dels vuitanta, l'actualització dels fitxers mestres no s'acostumava a fer en temps real (en línia). Els moviments d'actualització (altes, baixes i modificacions) produïts durant un cert interval de temps (dia, setmana, mes...) s'enregistraven en un fitxer de moviments associat a cada fitxer mestre, de manera que en finalitzar el corresponent interval s'actualitzava el fitxer mestre a partir del fitxer de moviments amb un procés especial. Una vegada el fitxer mestre estava actualitzat, el fitxer de moviments es creava de nou i es tornaven a enregistrar els moviments d'actualització produïts en el nou interval temporal. Aquest tipus de funcionament s'ha anomenat en temps diferit o en batch.

És evident que el funcionament en temps diferit porta problemes. Però en els anys setanta i vuitanta les comunicacions no permetien un altre funcionament. La revolució en el camp dels ordinadors personals, un cop se'n va abaixar el cost i van ser accessibles a tothom, va comportar el boom de les xarxes d'àrea local, que permetien el treball en temps real dels ordinadors de la xarxa. Quant als accessos remots, continuava necessitant-se el treball en temps diferit. El boom de les comunicacions permet oblidar el treball en batch, ja que amb les xarxes d'àrea estesa tothom té accés en temps real als fitxers mestres.

Podem oblidar del tot el treball en temps diferit? La resposta és no. En la vida quotidiana ens podem trobar en situacions en les quals el treball en temps diferit sigui necessari.

Penseu en els venedors o representants d'una gran empresa que tenen assignats els clients d'una determinada zona que han de visitar per a mostrar el catàleg dels productes i intentar aconseguir comandes de venda. Avui en dia és normal que els venedors portin tota la seva cartera de clients i el catàleg de productes en un ordinador portàtil amb una petita impressora, de manera que davant del client disposen de tota la informació. Ben segur que efectuaran altes, baixes i modificacions de clients i comandes de venda. Lògicament, aquests moviments els faran en el seu ordinador. Però a la casa del client el representant no pot estar treballant en temps real amb el sistema informàtic de l'empresa. Segurament el client disposarà de línia telefònica per a poder establir comunicació, però també és molt possible que no sigui d'alta velocitat i, a més, no és gens normal que el representant li demani la possibilitat d'utilitzar-la.

Per tant, s'estan enregistrant els moviments en un fitxer de transaccions i ens caldrà actualitzar els fitxers mestres de l'empresa. De la mateixa manera, a l'empresa s'estan produint moviments que han d'actualitzar els fitxers mestres de l'ordinador del representant. En algun moment (final del dia, final de la setmana...) cal actualitzar els fitxers mestres de l'ordinador portàtil i de l'empresa d'acord amb els fitxers de moviments generats a l'empresa i a l'ordinador portàtil respectivament. Aquest procés el podrà realitzar el representant establint connexió amb el sistema informàtic de l'empresa des de la xarxa d'àrea local o mitjançant connexió remota.

Funcionament actual en temps diferit

El funcionament en temps diferit és el que té lloc en un caixer automàtic quan no es té connexió amb la central bancària. L'usuari pot efectuar uns certs moviments, els qual queden enregistrats en un fitxer de moviments en el mateix caixer. Quan es restableix la connexió amb la central bancària, el saldo del compte bancari corresponent s'actualitza d'acord amb els moviments que han quedat enregistrats en el caixer automàtic.

c) Històrics

Són fitxers que contenen informació sobre situacions del passat. S'acostumen a utilitzar per a estadístiques. Alguns exemples són les factures d'un exercici econòmic, les nòmines dels empleats a final d'any...

Aquests fitxers mai no s'actualitzen. Només permeten consultes.

2) De moviments o transaccions

Són fitxers que contenen la informació per a actualitzar els fitxers mestres tal com hem explicat anteriorment.

3) De maniobra o temporals

Són fitxers de vida limitada i s'utilitzen com a auxiliars per a les feines com ara:

- connexions entre programes,
- processos d'ordenació sobre els fitxers,
- enregistrament de missatges d'error de processos batch.

1.7. Sistemes gestors de fitxers

Ja som coneixedors dels conceptes referents a la representació de la informació en els fitxers. Anem a veure diferents conceptes a tenir en compte en la gestió dels fitxers. Comencem presentant el concepte de sistema gestor de fitxers.

> Un sistema gestor de fitxers (SGF) és un conjunt de programes que conjuntament amb el sistema operatiu (en endavant SO) permet la gestió dels fitxers en un suport de memòria externa.

Ha de quedar molt clar que un SGF no deixa de ser un conjunt de programes que facilita la gestió dels fitxers. Ja sabem que el SO controla el funcionament del maquinari i del programari en execució. Per això és molt important tenir en compte que el SGF, com a conjunt de programes que és, és controlat pel SO de la màquina i, conjuntament amb aquest, dóna servei a les instruccions dels programes que accedeixen als fitxers.

El motiu de l'aparició dels SGF va ser l'intent d'aconseguir independitzar del SO i del maquinari la gestió dels fitxers en el desenvolupament de programes. (!!)

Heu de tenir molt clar l'objectiu que es pretenia aconseguir amb els SGF. En els subapartats anirem introduint diferents conceptes que ens permetran entendre el funcionament d'un sistema gestor de fitxers i la consecució de l'objectiu prefixat.

1.7.1. Fitxers intern i extern

En primer lloc cal distingir entre el fitxer extern i el fitxer intern.

El fitxer extern és el fitxer que està enregistrat en el suport de memòria externa. El fitxer intern és un objecte definit en el programa que vol accedir al fitxer extern.

Imaginem que tenim un fitxer de nom `CIUTATS.DAT` enregistrat en el directori `DADES` de la unitat lògica identificada per `D:` en un ordinador amb sistema operatiu Windows. Volem fer un programa que permeti accedir al fitxer `CIUTATS.DAT` per a efectuar-hi altes, baixes, modificacions i consultes. És a dir, un típic programa de gestió de fitxers.

El nostre programa (o conjunt de programes segons el disseny) ha d'accedir a un fitxer que es troba en un suport extern i que té un nom. En fer el programa, hem de tenir en compte els punts següents:

- El sistema operatiu en què estem treballant, ja que l'organització física dels fitxers és diferent en cada sistema operatiu (DOS, Windows, Linux...).

- El tipus de format del dispositiu físic (cilindres, cares, pistes, sectors..., en cas d'un disc; capacitat, densitat de gravació, pistes..., en cas d'una cinta).

- Les parts físiques del dispositiu en què hi ha enregistrat el fitxer i la manera com estan comunicades aquestes parts.

Fixeu-vos en la complicació de controlar totes aquestes característiques sense haver entrat encara en el tema que de veritat ens preocupa: poder gestionar ciutats del món!

El SGF, juntament amb el SO, es responsabilitza de la totalitat de controls anteriors, de manera que el programador només s'ha de preocupar de controlar el contingut dels registres a gestionar. Per a aconseguir això, el programador no fa referència, en els programes, al fitxer extern (`CIUTATS.DAT`, en aquest cas), sinó a un fitxer intern, objecte definit en el programa. L'equip SGF-SO, en temps d'execució, troba les instruccions de programa referents al fitxer intern, les quals executa sobre el fitxer extern. Evidentment, algú ha hagut de comunicar a l'equip SGF-SO quin fitxer extern cal mantenir enllaçat amb el fitxer intern que ha utilitzat el programador. Més endavant ja veurem qui és aquest algú.

Per a entendre millor el que estem dient, podem avançar que en pseudocodi farem quelcom semblant a:

```
tipus ciutat = tupla
                 nomPaís, nomCiutat : cadena [30];
                 superfície : real; /* Valor Nul = 0 */
                 nHabitants : enter; /* Valor Nul = -1 */
               fitupla
fitipus
var fciu: fitxer de ciutat; /* definició de fitxer intern */
    ...
fivar
```

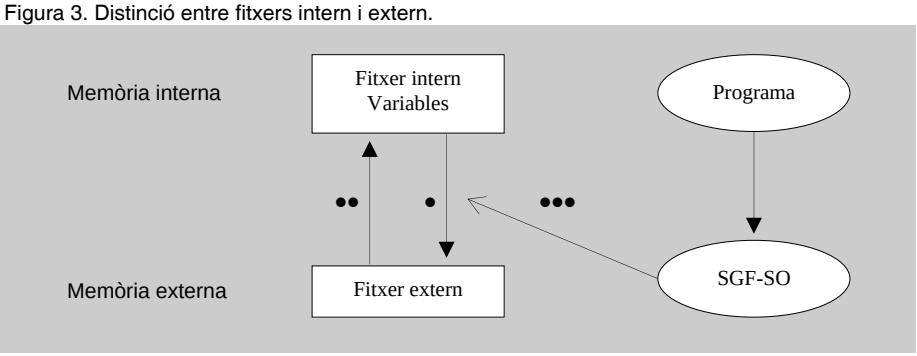
Fixeu-vos que s'ha definit la variable `fciu` com un tipus `fitxer`, desconegut fins al moment. Amb aquesta declaració s'obté l'objecte `fciu`, fitxer intern, que és el que s'utilitzarà en les instruccions del programa. Així, quan es vulgui llegir del fitxer i escriure al fitxer, de manera semblant a les instruccions "llegir de teclat" i "escriure per pantalla", tindrem instruccions per a accedir al fitxer que farem servir del tipus:

```
var r : ciutat; fivar
...
llegir_f (fciu, r); /* llegir del fitxer associat a fciu omplint la variable r */
...
escriure_f (fciu, r); /* escriure al fitxer associat a fciu el que hi ha a r */
```

És a dir, tota instrucció de programa per a accedir al fitxer utilitza el fitxer intern `fciu`. El SGF-SO coneix a quin fitxer extern està associat `fciu` i, per tant, totes les característiques necessàries per a l'accés, tant de lectura com d'escriptura (nom del fitxer extern, tipus de dispositiu, característiques d'enregistrament, capacitat del dispositiu...). No hem vist encara com podem informar el SGF-SO de l'associació entre els fitxers extern i intern.

La figura 3 pretén deixar clar el procés que permet la distinció entre fitxer extern i intern.

Figura 3. Distinció entre fitxers intern i extern.



Les instruccions d'accés a fitxers que inclou el programa les gestiona (•••) l'equip SGF-SO, de manera que:

- En cas d'instrucció de lectura (••) cerca la dada al fitxer extern i la deixa a la variable de memòria que correspongui, la qual serà tractada posteriorment pel programa.

- En cas d'instrucció d'escriptura (•), agafa la dada emmagatzemada pel programa a la corresponent variable de memòria i l'enregistra al fitxer extern (allà on pertoqui!).

1.7.2. Gestió d'entrades i sortides

L'esquema de la figura 3 és vàlid com a introducció al funcionament d'un SGF. La realitat acostuma a ser, però, un xic més complicada. Fixeu-vos que l'expressió acostuma a ser no té el mateix significat que l'expressió sempre és. És a dir, la situació esquematitzada és vàlida en certes ocasions.

El fet que possibilita la veracitat o la falsedat de l'esquema anterior és la no utilització o la utilització, respectivament, de memòria intermèdia (buffers) per a la transferència de les dades entre el fitxer extern i les variables de programa.

Recordeu la utilització de la memòria intermèdia (buffer) en certes instruccions de lectura per teclat?

> Un buffer de fitxer és una part de la memòria principal destinada a emmagatzemar temporalment les dades que s'han de transferir de la memòria interna a la memòria externa i viceversa.

Les operacions d'entrada-sortida (E/S) en fitxers són molt lentes (per més ràpids que siguin els actuals dispositius de memòria externa) en comparació amb la velocitat de procés de la CPU. Per aquest motiu, és convenient reduir al màxim aquestes operacions. Això s'aconsegueix fent servir buffers.

Podem gestionar les E/S en fitxers utilitzant o no utilitzant buffers. Aprofundirem en les dues possibilitats.

1) Gestió d'E/S en fitxers sense utilitzar buffers

En aquest cas, l'esquema de la figura 3 és correcte. Quan el SO troba una instrucció E/S sobre el fitxer intern, el SGF entra en acció i efectua l'E/S corresponent sobre el fitxer extern. És a dir, en aquest cas, una E/S lògica provoca sempre una E/S física.

Aquesta situació és similar a la que es produeix en l'àmbit domèstic quan l'encarregat d'anar a comprar va adquirint els aliments a mesura que els necessita. Estareu d'acord que anar a comprar és una pèrdua de temps i que cal minimitzar-ho al màxim, oi? De segur que l'encarregat d'anar a comprar guanyaria temps si omplís la nevera i el rebost i anés agafant els aliments a mesura que els necessités. És a dir, si es poden fer servir la nevera i el rebost es guanya en eficiència, oi? La nevera i el rebost són els equivalents dels buffers en la gestió d'E/S en fitxers.

2) Gestió d'E/S en fitxers utilitzant buffers

Quant a la utilització de buffers cal distingir els conceptes següents:

Una E/S lògica és el resultat d'una instrucció de lectura/escriptura en un programa i consisteix en la transferència d'un registre entre buffer i variable de programa.

Una E/S física és el resultat d'una instrucció de lectura/escriptura en un programa i consisteix en la transferència d'un bloc entre suport físic i buffer.

Un registre o registre lògic és la dada transferida entre buffer i variable de programa en una E/S lògica i coincideix amb la percepció que es té dels registres que conté el fitxer.

Un bloc o registre físic o pàgina és la unitat de transferència entre buffer i suport físic que es produeix en una E/S física. La seva grandària acostuma a dependre de l'equip SGF-SO.

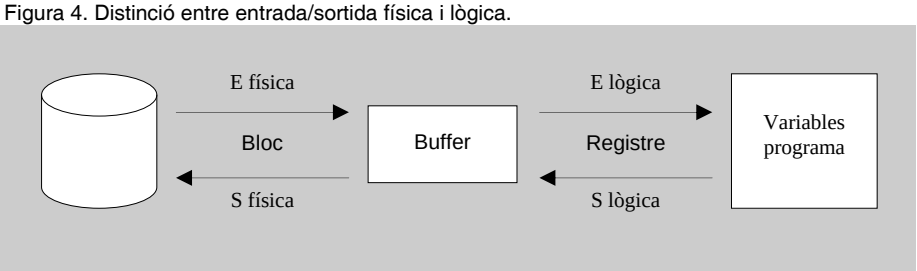
El factor de bloqueig és el nombre de registres que conté un bloc.

Esquemàticament, ho podem representar amb la figura 4.

En certs SGF, és l'usuari, és a dir, el programa, qui defineix la grandària de buffer i de bloc. Un registre no pot estar a cavall de diversos blocs. Per tant, interessa ajustar la grandària de bloc perquè contingui un nombre

exacte de registres. Hi ha SGF que permeten diferents grandàries de bloc i de buffer. En aquests casos, cal escollir la grandària de bloc que més s'ajusti a la grandària dels registres a tractar. És possible que en els blocs quedi espai no utilitzat. Cal minimitzar-lo al màxim.

Figura 4. Distinció entre entrada/sortida física i lògica.



Recordeu que ens falta veure com l'equip SGF-SO coneix quin fitxer extern està associat a un fitxer intern. En algun moment s'ha de produir el lligam i en algun altre moment s'ha d'eliminar. Ho veurem amb deteniment més endavant. Ara, però, heu de saber que l'equip SGF-SO destina un o diversos buffers a cada lligam fitxer extern-fitxer intern. Mentre no hi ha lligam, no hi ha buffers assignats.

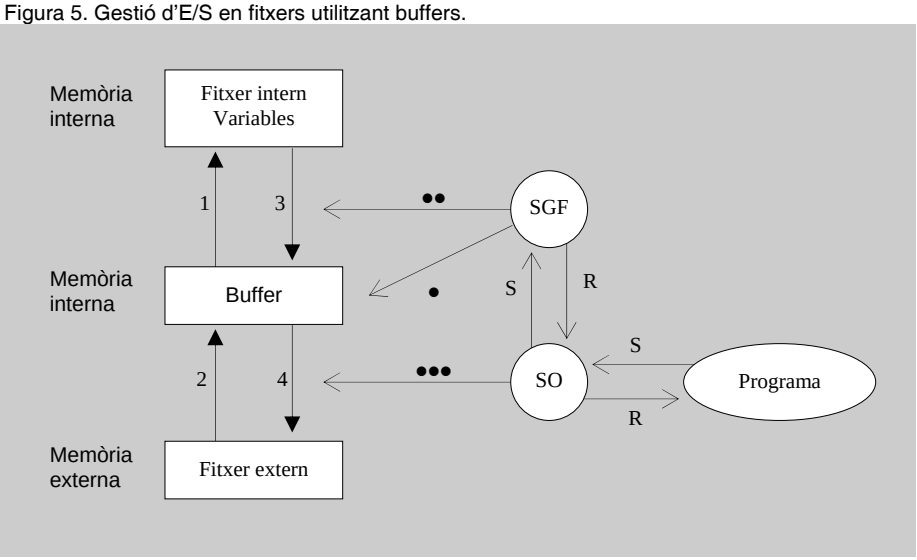
La figura 5 ens mostrea l'esquema de funcionament d'un SGF amb la utilització de buffers.

Per tal d'entendre el funcionament vegeu la simulació de l'execució d'una instrucció de lectura i d'una instrucció d'escriptura.

a) Simulació d'execució d'una instrucció d'entrada

Quan el SO troba en el programa una sol·licitud (S) de lectura sobre fitxers la traspassa a al SGF, el qual fa una comprovació (•) sobre l'existència en el(s) buffer(s) de la dada cercada. Podria ser-hi? Ja veureu més endavant que sí.

Figura 5. Gestió d'E/S en fitxers utilitzant buffers.



Si hi és, al SGF controla (••) el traspàs (1) de la dada des del buffer cap a les variables de programa. S'ha executat una entrada lògica, consistent a traspassar un registre des de buffer cap a variable de programa. La dada traspassada s'acostuma a anomenar registre o registre lògic.

Si en la comprovació (•) al SGF comprova que la dada cercada no és en el buffer, cosa que, per exemple, sempre succeirà en la primera instrucció de lectura, ho comunica (R) al SO, cosa que provoca que aquest controli (•••) una lectura física sobre el fitxer extern per traspassar (2) un bloc cap al buffer. Recordeu que un bloc és la unitat de transferència entre memòria externa i memòria interna.

Cada bloc traspassat des del fitxer extern cap al buffer és una entrada física. Segons siguin la grandària del bloc (cosa que depèn del SO) i la grandària del registre del fitxer intern, en un bloc hi pot haver un o diversos registres. Per tant, una entrada física pot situar diferents registres lògics en el buffer, cosa que dóna sentit a l'actuació inicial de al SGF davant una lectura lògica (comprovació de la possibilitat de trobar el registre cercat en el buffer).

Una vegada el SO ha acabat la gestió de la transferència (2) del bloc cap al buffer, ho comunica (S) al SGF, el qual controla (••) la transferència (1) del registre que correspongui des del buffer cap a la variable registre del programa.

Per a acabar el procés, el SGF comunica (R) al programa, per mitjà del SO, el resultat de l'operació, el qual ha pogut ser satisfactori o no. El programa ha d'haver estat dissenyat de manera que pugui actuar en conseqüència segons el resultat comunicat pel SGF.

b) Simulació d'execució d'una instrucció de sortida

Quan el SO troba en el programa una sol·licitud (S) d'escriptura sobre fitxers la traspassa al SGF, el qual controla (••) el traspàs (3) del registre que s'ha d'escriure al buffer, on hi ha d'haver el bloc contenidor del registre. S'ha efectuat una sortida lògica.

Posteriorment, el SGF comunica (R) al SO que ja ha acabat la seva feina, cosa que provoca que el SO controli (•••) el traspàs (4) del bloc des del buffer cap al fitxer extern. En aquest moment s'està produint una sortida física.

Hi ha SGF que permeten la possibilitat de no efectuar la sortida física immediatament després de la sortida lògica, per tal de guanyar en

velocitat, ja que és molt probable que a continuació s'hagi de tornar a enregistrar el mateix bloc a causa d'una nova sortida lògica que provoca registres que han d'estar en el mateix bloc. En aquest cas, el SO no efectua l'enregistrament (4) fins que el SGF en dóna l'autorització corresponent. Aquest funcionament permet millorar la velocitat de procés, però és perillós, ja que si l'execució és avortada per qualsevol motiu es queden blocs pendents de ser enregistrats en els buffers i si feia estona que el programa havia donat ordre d'escriptura...

Per tal d'acabar el procés, el SGF comunica (R) al programa, per mitjà del SO, el resultat de l'operació, el qual ha pogut ser satisfactori o no. El programa ha d'haver estat dissenyat de manera que pugui actuar en conseqüència segons el resultat comunicat pel SGF.

1.7.3. Estructura física dels fitxers

En desenvolupar programes sobre fitxers, l'analista programador els considera com un conjunt organitzat de registres als quals té accés. Aquests registres són, però, dades virtuals, ja que en el programa s'hi fa referència com si físicament existissin en aquell format, quan la realitat és que en el suport físic existeixen d'una manera molt diferent. L'equip SGF-SO és l'encarregat de proporcionar-les en el format esperat. De la mateixa manera, des del programa no s'és conscient de com les dades es troben en el fitxer físic. Semblen no existir, però existeixen. Són dades transparents per a l'analista programador.

El nostre objectiu és desenvolupar aplicacions informàtiques que gestionin fitxers; per tant, ens interessa conèixer a fons les dades des del punt de vista lògic, és a dir, des del vessant de les dades virtuals. No obstant això, és interessant tenir consciència de com és l'estructura física dels fitxers.

Un fitxer físic és palpable, és a dir, se'n pot comprovar l'existència observant el suport físic en el qual està emmagatzemat i, fins i tot, se'n pot veure el contingut amb alguna ordre del SO corresponent. No obstant això, no us estranyeu si utilitzant una determinada ordre del SO no veieu altra cosa que símbols estranys. Només en podreu veure bé el contingut si aquest ha estat enregistrat en format ASCII, cosa no gaire normal.

Ordres per a visualitzar un fitxer

Recordeu les ordres dels sistemes operatius DOS-Windows i Unix-Linux per a visualitzar el contingut d'un fitxer?

DOS-Windows: `type <nom>`
Unix-Linux: `cat <nom>`

Els fitxers tenen una estructura física gestionada per l'equip SGF-SO. Pel fet de ser un conjunt de registres, l'estructura física d'un fitxer

dependrà molt de l'estructura física dels seus registres. De la mateixa manera, pel fet que un registre és format per diversos camps, l'estructura física del registre dependrà de l'estructura física dels seus camps.

Fem un ràpid repàs sobre alguns conceptes relatius a l'estructura física dels camps, dels registres i, en definitiva, dels fitxers.

1) Estructura física dels camps

Recordem que havíem definit el concepte de domini com el conjunt de valors legals que pot prendre un atribut. Podem tenir dominis del tipus "conjunt de naturals de 6 dígits", "conjunt de reals positius i no superiors a 1000", "conjunt de cadenes de longitud 30 caràcters", "conjunt de cadenes de longitud no superior a 15 caràcters"...

Els valors concrets que pren un atribut estan representats en el camp corresponent i han d'estar enregistrats físicament en el suport. Podem distingir entre un enregistrament implícit i un enregistrament explícit.

> L'enregistrament implícit d'un camp té lloc quan és l'equip SGF-SO el que determina el nombre d'octets necessaris en funció del tipus de dada del camp. L'enregistrament explícit es dóna quan és el programador el qui determina la manera com s'ha d'enregistrar físicament el camp.

Així, un enregistrament implícit té lloc quan l'equip SGF-SO decideix emmagatzemar un valor enter de quatre octets utilitzant exactament els quatre octets que ocupa en memòria. Un enregistrament explícit es dóna, en el mateix cas, quan s'enregistra el valor numèric tal com acostumem a visualitzar-lo en pantalla o en suport paper.

Exemple

Suposeu en llenguatge C la variable `unsigned int n` emmagatzemant el valor 50000.

L'enregistrament implícit en un fitxer gravaria els dos octets que ocupa la variable n (suposant plataforma de 16 bits en la que els `int` ocupen dos octets). Si observéssiu el contingut del fitxer veuríeu la seqüència dels dos caràcters ASCII corresponents als dos octets enregistrats. Els dos octets corresponen als valors 80 i 195. El primer caràcter és la lletra P, mentre que el segon, com que és superior a 127, correspon a un caràcter que dependrà de la taula ASCII activa. Sabeu per què són els valors 80 i 195, oi?

En canvi, un enregistrament explícit es produeix quan el programador decideix que enregistrarà el valor de n utilitzant el format "%6u". Si visualitzéssiu el contingut del fitxer us trobaríeu un espai en blanc seguit del valor 50000. Sabeu per què, oi?

D'altra banda, cal distingir si el camp serà de longitud fixa o de longitud variable.

> Un camp és de longitud fixa quan en el suport físic sempre ocupa el mateix nombre d'octets, sigui quin sigui el valor representat. En canvi, un camp és de longitud variable quan el nombre d'octets utilitzats depèn del valor enregistrat.

Així, si en un enregistrament explícit el programador decideix gravar el valor d'una longitud o d'una altra en funció del valor, tenim un camp de longitud variable.

Exemple

Suposem en llenguatge C la variable `char s[45]` emmagatzemant el valor "HOLA".

El programador pot decidir enregistrar els 45 octets que ocupa la variable. En aquest cas, un camp d'aquest tipus tindria una longitud fixa de 45 octets, fos quin fos el valor emmagatzemat.

Però el programador pot decidir enregistrar només els octets amb significat. Llavors el valor a enregistrar ocupa `strlen(s)` o `strlen(s) + 1`.

Que un camp sigui de longitud fixa o variable no està lligat amb el fet d'haver-se produït un enregistrament explícit. En els enregistraments implícits, l'equip SGF-SO també pot prendre decisions respecte a la longitud fixa o variable dels camps.

2) Estructura física dels registres

L'estructura física dels registres és donada per l'estructura física dels camps que els componen. Així ens podem trobar:

- Registres de longitud fixa: tots els registres del fitxer tenen igual grandària.

  En aquest cas, qualsevol registre del fitxer ha de ser format pels mateixos camps de longitud fixa. Però pot haver-hi registres amb diferent estructura de camps? Evidentment! Ho comentem a continuació.

- Registres de longitud variable: cada registre té la seva pròpia longitud.

  No tots els SGF proporcionen aquesta possibilitat. La longitud variable pot ser motivada per:

– Camps de longitud variable.

– Camps opcionals que provoquen diferent estructura de registre.

  Recordeu que per a l'atribut quants parts ha tingut una persona comentàvem que no era aplicable al sexe masculí i justificàvem l'existència del valor nul?

  Podríem optar per no enregistrar els valors nuls motivats per atributs no aplicables en funció del valor d'un altre atribut. Així, en funció del valor del camp sexe, existent en qualsevol registre, podríem tenir o no el camp nombre de parts.

– Nombre variable de repeticions d'un camp.

  Recordeu que per a l'atribut quants idiomes coneix comentàvem que el nombre de valors possible no era fix? Havíem comentat que la millor solució era tenir un altre fitxer auxiliar on poguéssim enregistrar els idiomes que coneix cada persona. Ara bé, podríem optar –si al SGF ho permet– per un camp amb un nombre variable de repeticions, tantes com idiomes coneix la persona.

1.7.4. Operacions sobre fitxers

Estudiarem ara les operacions que seria interessant poder efectuar sobre fitxers. Veurem que els SGF en proporcionen algunes, però no totes.

1) A nivell de camp

Les operacions que se'ns ocorren són:

- Afegir un nou camp.
- Suprimir un camp.
- Modificar el domini de valors d'un camp.

Els SGF no proporcionen aquestes operacions. En canvi, els sistemes gestors de bases de dades (SGBD) sí que les proporcionen.

Què hem de fer si, per necessitats de gestió, cal efectuar canvis en l'estructura dels registres en algun fitxer? No hi haurà altre remei que crear, d'una banda, un fitxer nou amb una nova estructura i, de l'altra, un programa que copiï tots els registres del fitxer original en el nou fitxer amb les transformacions, pel que fa al camp, que siguin necessàries. Posteriorment, el fitxer original s'esborra i el nou fitxer rep el nom que tenia el fitxer original. És a dir, quan el sistema gestor

de fitxers no proporciona la funcionalitat requerida entra en acció el programador, que desenvolupa el programa adequat.

Existeix, però, algun SGF que incorpora alguna eina per a efectuar aquest tipus d'operacions i estalviar la feina de programació. En aquest cas, hem de considerar que el SGF ha començat a avançar els seus passos cap al que és un SGBD.

2) A nivell de registre

Recordeu que les E/S lògiques es localitzen en l'àmbit del registre. És a dir, si es vol tenir el valor d'un o diversos camps, cal llegir tot el registre. Si es vol modificar el valor d'un camp, cal reescriure tot el registre. Les operacions que se'ns ocorren són:

- Fer consultes, és a dir, lectures.

- Fer actualitzacions, és a dir, altes, baixes i modificacions (de valors emmagatzemats en els camps, no pas del domini dels camps).

Els SGF acostumen a incorporar aquestes operacions. En cas que no les incorporessin, seria feina del programador efectuar-ne la implementació corresponent. Per si ens pertoca implementar-les cal tenir present que:

- Tenim dues maneres d'efectuar altes de registres:

– Afegint-les pel final del fitxer.
– Inserint-les per entremig dels registres existents.

- Tenim dues maneres d'efectuar baixes de registres:

– Marcant l'espai ocupat pel registre que cal donar de baixa amb un senyal que indiqui que està de baixa. Si s'actua d'aquesta manera, s'ha de tenir en compte aquest senyal en efectuar les altes, les consultes i les modificacions, ja que en les altes podem aprofitar els espais marcats per les baixes i en les consultes i les modificacions cal que aprofitem els registres marcats com a baixes. Si es produeixen moltes baixes ens podem trobar un fitxer amb molts forats, cosa que faria necessari un procés de compactació per a recuperar l'espai esborrat.

– Compactant l'espai ocupat pel registre que cal esborrar en cada baixa.

- Tenim dues maneres d'efectuar modificacions dels valors dels camps:

– Aixafant el registre antic, és a dir, reescrivint-lo sobre el mateix espai.

– Esborrant el registre que cal modificar i donant-lo d'alta posteriorment amb els valors actualitzats dels camps.

3) A nivell de fitxer

Les operacions que se'ns ocorren són:

- Crear físicament el fitxer, cosa que ha d'efectuar el SGF.

- Esborrar físicament el fitxer, cosa que algun SGF gestiona, però que en la majoria dels casos acostuma a fer-se des del SO.

- Consultar i actualitzar diversos registres a la vegada, operacions que no poden fer els SGF i sí que poden fer els SGBD.

- Classificar o ordenar el fitxer, és a dir, resituar els registres de tal forma que quedin ordenats per algun(s) camp(s) determinat(s). La majoria de SGF no tenen aquesta funcionalitat. Si necessitem ordenar un fitxer haurem d'efectuar el programa corresponent.

4) A nivell de diversos fitxers

Les operacions que se'ns ocorren són:

- Fusionar diversos fitxers en un, cosa que pot passar quan cal agrupar els fitxers de clients, proveïdors, factures... en una fusió d'empreses.

- Partir un fitxer en diversos fitxers, segons les característiques d'algun(s) camp(s).

- Actualitzar amb còpia, és a dir, generar un nou fitxer mestre a partir del fitxer mestre existent i d'un fitxer de moviments.

Aquestes operacions no estan disponibles en els SGF. Per tant, serà feina del programador dissenyar els programes corresponents.

1.7.5. Accés als fitxers interns

Recordem que el nostre objectiu principal és desenvolupar programes que gestionin fitxers. Sabem que, en un programa, qualsevol operació sobre un fitxer s'efectua sobre un fitxer intern que està enllaçat amb un fitxer extern (encara no sabem, però, com s'efectua l'enllaç). Ara

ens interessa conèixer els mecanismes que al SGF posa a les nostres mans per a accedir a la informació.

Hi ha dues maneres d'accedir a la informació:

a) Mitjançant un accés seqüencial. Per a accedir a un registre cal passar obligatòriament pels anteriors, des de l'inici.

b) Mitjançant un accés directe. Per a accedir a un registre no cal passar pels anteriors.

Fixeu-vos que en aquestes definicions apareix l'expressió "pels anteriors", que implica l'existència d'algun tipus de classificació entre els registres. Tenim dues maneres de classificar els registres:

a) Segons la posició física que ocupen dins el fitxer.
b) Segons el valor d'un o diversos camps per als quals ha d'existir algun tipus d'ordenació.

Si combinem els dos tipus d'accés amb els dos tipus de classificació obtenim quatre possibilitats.

Suposem el fitxer representat a la taula 7. Observeu que es tracta d'un fitxer que té dos camps: nom i edat. Suposem que en aquests moments hi ha quatre registres. Les referències R1, R2, R3 i R4 indiquen la posició física que ocupen (primer, segon, tercer i quart).

Taula 7. Exemple de fitxer

	Nom	Edat
R1	Pep	17
R2	Lluís	21
R3	Joan	25
R4	Maria	40

- Suposem que es vol accedir al registre que està a la quarta posició. Cal, doncs, efectuar un accés seqüencial o directe per posició.

– En cas d'un accés seqüencial per posició, per a accedir a un registre cal haver passat pels registres anteriors. Per tant, per a arribar al registre R4 cal haver passat pels registres R1, R2 i R3.

– En cas d'un accés directe per posició, podem accedir directament a un registre sense haver de passar pels registres que ocupen una posició anterior. Per tant, accediríem directament al registre R4.

• Suposem que es vol cercar un registre que tingui per nom "Maria".

– Si el fitxer no disposa de cap classificació per al camp nom, no hi haurà altre remei que utilitzar l'accés seqüencial per posició i, començant per l'inici, anar recorrent els registres i comprovant que cadascun té el valor "Maria" per al camp nom. Per tant, passarem pels registres R1, R2, R3 i R4 i en aquest últim trobarem el valor cercat.

– Si el fitxer disposa d'accés seqüencial per al valor del camp nom, podrem utilitzar-lo, de manera que, començant pel primer registre segons el camp nom i seguint pels registres següents, anirem comprovant si trobem un registre amb el valor cercat. Així passarem pels registres R3 (Joan), R2 (Lluís) i R4 (Maria) on, en trobar-se el valor cercat, s'acaba el recorregut.

– Si el fitxer disposa d'accés directe per al valor del camp nom, podrem utilitzar-lo, de manera que podem sol·licitar directament el registre que tingui el valor "Maria" per al camp nom. El SGF ens comunicarà que existeix aquest registre i ens el subministrarà.

• Suposem que es vol cercar un registre que tingui per nom "Lídia".

– Si el fitxer no disposa de cap classificació per al camp nom, no hi haurà altre remei que utilitzar l'accés seqüencial per posició i, començant per l'inici, anar recorrent els registres i comprovant que cadascun té el valor "Lídia" per al camp nom. Per tant, passarem pels registres R1, R2, R3 i R4 sense trobar el valor cercat.

– Si el fitxer disposa d'accés seqüencial per al valor del camp nom, podrem utilitzar-lo, de manera que, començant pel primer registre segons el camp nom i seguint pels registres següents, anirem comprovant si trobem un registre amb el valor cercat. Així passarem pels registres R3 (Joan) i R2 (Lluís). No cal continuar, ja que sense haver trobat el valor "Lídia" hem arribat a un registre que té un valor superior ("Lluís").

– Si el fitxer disposa d'accés directe per al valor del camp nom, podrem utilitzar-lo, de manera que podem demanar directament el registre que tingui el valor "Lídia" per al camp nom. El SGF ens comunicarà que no existeix cap registre amb aquest valor.

Fixeu-vos que la utilització d'accés segons el valor implica l'existència d'una via d'accés, la qual pot ser constituïda per un o diversos camps. En l'exemple anterior, hem estat treballant amb la via d'accés formada pel camp nom. Podríem parlar d'una altra via d'accés constituïda pel camp edat. Però també podem tenir vies d'accés constituïdes per diferents camps, com en el cas de nom + edat o edat + nom. En aquests casos, s'accedeix als registres classificats pel primer camp que forma la via d'accés, i en cas d'igualtat de valor, es té en compte el(s) camp(s) següent(s). (!!)

1.7.6. Tipificació de fitxers interns

Tenim quatre maneres d'accedir a la informació emmagatzemada en fitxers. Un SGF pot oferir qualsevol combinació de les quatre possibilitats. Per tant, quants tipus diferents de SGF podeu trobar?
La resposta és 15. Aquesta solució s'obté sumant $C_{4,1} + C_{4,2} + C_{4,3} + C_{4,4}$.

$C_{4,1}$: Combinacions de 4 possibilitats agafant-ne 1 = 4
$C_{4,2}$: Combinacions de 4 possibilitats agafant-ne 2 = 6
$C_{4,3}$: Combinacions de 4 possibilitats agafant-ne 3 = 4
$C_{4,4}$: Combinacions de 4 possibilitats agafant-ne 4 = 1

És a dir, tenim potencialment 15 tipus de SGF. Comercialment, però, n'hi ha tres tipus principals, que tot i ser SGF s'anomenen simplement fitxers: (!!)

a) Fitxers seqüencials, que permeten l'accés seqüencial segons la posició.

b) Fitxers relatius, que permeten l'accés seqüencial i directe segons la posició.

c) Fitxers seqüencials indexats, que permeten l'accés seqüencial i directe segons el valor amb la possibilitat de diverses vies d'accés.

Altres dos tipus de SGF que cal destacar són: els fitxers calculats i els fitxers de text.

1) Fitxers calculats, també anomenats aleatoris o hashing, els quals no estan comercialitzats i que s'implementen amb la utilització de fitxers relatius. Permeten l'accés seqüencial i directe segons la posició (ja que es basen en fitxers relatius) i l'accés directe segons el valor d'una única via d'accés. En moltes ocasions se'ls considera com una especificitat de fitxers relatius.

2) Fitxers de text, que són una espècie de fitxers seqüencials que fan servir caràcters especials com fi_de_línia, fi_de_pàgina... Aquests fitxers s'utilitzen per a enregistrar informació sense seguir cap tipus d'estructura mitjançant registres i camps.

1.7.7. Lligams entre fitxers intern i extern

Argot intormàtic

El terme anglès emprat en informàtica per a fer referència al lligam entre el fitxer intern i el fitxer extern és binding. Sovint s'utilitza directament aquest terme.

Ha arribat el moment de veure com s'estableix el lligam entre el fitxer extern i el fitxer intern. Aquest lligam s'ha de dur a terme obligatòriament abans de fer cap tipus d'operació amb el fitxer intern.

Ara bé, per a poder fer el lligam és necessària l'existència del fitxer extern. Hi ha dues maneres de crear els fitxers externs:

- Amb una instrucció de programa que executarà el SGF.

- Amb una eina executable des del sistema operatiu, la qual no deixa d'efectuar una crida al SGF perquè creï el fitxer de la mateixa manera que el crea a partir de l'execució d'una instrucció del programa.

La manera més utilitzada per a establir el lligam entre el fitxer extern i el fitxer intern és mitjançant una instrucció de programa anomenada instrucció d'obertura del fitxer. (!!)
Acostuma a ser una instrucció `obrir_fitxer` que s'utilitza com es veu en el següent tros de codi:

```
tipus T = tupla
          ...
          fitupla
fitipus
var f: fitxer de T;
    error: ???;
    ...
fivar
...
error = obrir_fitxer (f, "FITXER_EXTERN", ...);
...
```

Amb la funció `obrir_fitxer` s'estableix el lligam entre el fitxer intern f i el fitxer extern de nom `FITXER_EXTERN`. Observem que aquesta funció pot tenir més arguments, els quals poden utilitzar-se per a:

– crear el fitxer extern si no existeix,

– decidir si al fitxer extern s'hi podrà accedir només per a lectura o per a la lectura i l'escriptura.

Així mateix, la funció `obrir_fitxer` ens retorna un `error` que ens indica si el SGF ha pogut o no establir el lligam. Normalment, en cas d'error, retorna diferents valors que ens permeten esbrinar el motiu del problema. En aquest exemple hem plantejat la instrucció `obrir_fitxer` com una funció. En alguns SGF de fitxers pot consistir en una acció. D'alguna manera sempre hi ha la possibilitat de conèixer si hi ha hagut algun error i, en cas afirmatiu, esbrinar quin és l'error que s'ha produït.

I si el programa anterior es vol executar sobre un fitxer diferent del que hem escrit en la instrucció d'obertura? No penseu que cal canviar el codi del programa i obtenir un nou executable. El paràmetre que fa referència al fitxer extern pot ser una variable tal que el valor s'ompli amb una instrucció entrada per teclat o a partir dels arguments en la línia d'execució del programa.

El lligam es manté fins que es trenca, cosa que acostuma a produir-se mitjançant una altra instrucció semblant a: ‼

```
error = tancar_fitxer (f);
```

En el moment d'establir el lligam, el SO assigna el(s) buffer(s) necessari(s) (si es treballa amb buffer) i altres recursos que estan assignats al lligam fitxer intern-extern mentre aquest existeix. En el moment de trencar-se el lligam, els recursos assignats s'alliberen.

És a dir, els lligams consumeixen recursos. Per tant, no és un bon mètode de treball establir tots els lligams necessaris al llarg del programa a l'inici d'execució i alliberar-los en acabar l'execució si algun dels lligams es necessita en una zona molt específica. És a dir, cal establir els lligams quan sigui necessari, ja que consumeixen recursos del SO.

Quins tipus de lligams es poden establir?

- 1 fitxer intern associat a 1 fitxer extern al llarg del programa.

- 1 fitxer intern associat a n fitxers externs al llarg del programa però no simultàniament.

- n fitxers interns associats a 1 fitxer extern. Si no estan associats simultàniament, no hi ha cap problema. Si, per contra, estan associats simultàniament, hi pot haver problemes de concurrència (si no hi ha un bon control) ja que per diferents canals (fitxers interns) s'accedeix a la mateixa informació i es poden efectuar operacions contradictòries, com per exemple estar esborrant un registre a través

d'un fitxer intern mentre que per un altre fitxer intern s'intenta modificar el mateix registre.

- n fitxers interns associats a m fitxers externs, tenint en compte les consideracions dels tres apartats anteriors.

1.7.8. Independència física de les dades

El motiu de l'aparició dels SGF va ser l'intent d'aconseguir independitzar del SO i del maquinari la gestió dels fitxers en el desenvolupament de programes, fet que es coneix com independència física de les dades.

> S'aconsegueix una independència física de les dades quan la gestió de les dades no depèn del sistema operatiu ni del maquinari.

A partir d'ara desenvoluparem programes que gestionen fitxers. L'única dada referent als fitxers externs que haurem de tenir en compte és el nom i el dispositiu (unitat lògica, directori...) pel qual s'hi ha d'accedir. En cap moment haurem de preocupar-nos de si es tracta d'un disc dur, un disc flexible, un disc òptic o una cinta. Tampoc no ens haurem de preocupar de la grandària, el nombre de cares, pistes, sectors... Per tant, la gestió de les dades és independent de la seva organització física. (!!)

> En els SGF la independència física de les dades s'aconsegueix fent la distinció entre fitxer intern i fitxer extern i deixant que sigui el SGF qui s'encarregui conjuntament amb el SO de la gestió física de les dades.

2. Gestió de fitxers via el llenguatge C

El llenguatge C incorpora la possibilitat de gestionar els fitxers del sistema de fitxers del sistema operatiu i ho permet amb la utilització de buffers i sense. En un inici, com que era un llenguatge que evolucionava de la mà del sistema operatiu Unix, que és un sistema operatiu d'E/S sense buffer, les primeres funcions que va incorporar per a gestionar E/S s'acomodaven a les necessitats del SO Unix, és a dir, sense buffer. L'evolució de la gestió dels fitxers ha provocat, però, que l'actual estàndard ANSI C no incorpori les funcions d'E/S sense buffer tot i que es conserva en la majoria d'implementacions de llenguatge C per tal de ser compatible amb versions anteriors.

En aquest apartat estudiarem amb deteniment les funcions que l'estàndard ANSI C incorpora per a la gestió de fitxers. Totes aquestes funcions depenen de la utilització de buffers.

2.1. Streams

En el llenguatge C és molt important conèixer el concepte stream, traduïble per corrent o flux.

> Un stream és l'objecte que el llenguatge C incorpora perquè es pugui utilitzar qualsevol dispositiu.

Fixeu-vos que és l'stream allò que permet utilitzar qualsevol dispositiu, és a dir, unitats de disc o cinta, ports, impressora, terminals.

Un stream és com una generalització del concepte de fitxer intern. D'igual manera que un fitxer intern és l'objecte teòric que ha d'utilitzar el programador per a poder fer referència a un fitxer extern, el programador en llenguatge C farà servir obligatòriament els streams per a poder fer referència als dispositius que s'han de gestionar (entre els quals els fitxers externs).

Quan s'executa un programa, C obre de manera automàtica tres streams associats a dispositius:

- `stdin`: dispositiu d'entrada estàndard (teclat),

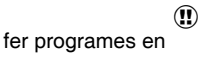
- `stdout`: dispositiu de sortida estàndard (pantalla),

- `stderr`: dispositiu d'errors estàndard (pantalla).

En el SO DOS, a més d'aquests tres streams n'hi pot haver, segons la configuració de la màquina, dos més:

- `stdaux`: dispositiu auxiliar estàndard (port sèrie),

- `stdprn`: dispositiu d'impressió estàndard (impressora paral·lel).



Abans de fer programes en Turbo C que pretenguin imprimir en impressores en Windows 2000/XP/Vista, llegiu l'annex "Impressió per `stdprn` en aplicacions DOS des de Windows" de la web d'aquest crèdit.

En el llenguatge C, com que tots els dispositius es gestionen per streams, les funcions genèriques que permetin treballar amb streams ens facilitaran la gestió dels fitxers en dispositius de memòria externa, la impressora, la pantalla i el teclat.

Els streams són punters al tipus de dada `FILE` aportada pel llenguatge C. Podem utilitzar els streams automàticament oberts pel llenguatge en qualsevol funció que requereixi com a argument un punter a una estructura de tipus `FILE`.

El tipus `FILE` és un tipus definit al fitxer `stdio.h`. Si el consulteu, podreu veure que consisteix en una estructura que té diverses entrades. No totes les implementacions de llenguatge C utilitzen la mateixa definició, tot i que totes assoleixen el mateix objectiu. Entre els diferents elements que formen l'estructura cal destacar l'existència de:

- una màscara (flags) que conté el mode d'accés al fitxer i els errors que es produeixen en accedir-hi,
- uns punters al buffer d'E/S i al següent element dins el buffer,
- un descriptor del fitxer, que no és altra cosa que un valor enter que identifica cada fitxer.

L'accés a un fitxer mitjançant la utilització d'un fitxer intern és un accés d'alt nivell. Quan s'obre un fitxer, al SGF li assigna un descriptor de fitxer (queda emmagatzemat en l'estructura FILE apuntada per l'stream corresponent). El llenguatge C aporta funcions per a accedir als fitxers a través del seu descriptor: accés de baix nivell. Nosaltres no treballarem a baix nivell, per la qual cosa no farem més referència als descriptors dels fitxers, però heu de conèixer-ne l'existència ja que en els manuals de referència del llenguatge C i en els sistemes d'ajuda trobareu funcions que en faran esment. Només un darrer detall: els descriptors associats als streams predefinits són 0 per a `stdin`, 1 per a `stdout`, 2 per a `stderr`, 3 per a `stdaux` i 4 per a `stdprn`.

Així doncs, per a treballar amb un fitxer en memòria externa haurem de definir un stream f, de la manera següent:

```
#include <stdio.h>
FILE *f;
```

Aquesta definició equival a la declaració en pseudocodi d'un fitxer intern:

```
f: fitxer_seq de T;
```

Observeu que a diferència del pseudocodi, en el qual cal definir el tipus `T` dels registres que emmagatzema el fitxer, el llenguatge C no en fa cap esment. Això és motivat perquè al SGF que incorpora el llenguatge C és molt rudimentari i deixa tota la feina de tractament de registres a càrrec del programador. Per aquest motiu, el llenguatge C no ha estat un llenguatge utilitzat normalment en aplicacions de gestió en les quals la manipulació de fitxers és constant. Nosaltres farem servir el SGF que incorpora el llenguatge C per a treballar amb fitxers seqüencials i amb fitxers relatius. Aquests dos tipus de SGF donen unes baixes prestacions que serem capaços d'implementar fàcilment en el llenguatge C. Per contra, els SGF indexats donen unes prestacions molt altes, i no té sentit dedicar esforços a implementar-les en el llenguatge C quan en el mercat es poden trobar SGF que permeten vincular-los amb el llenguatge C. Però això ja ho veurem més endavant.

2.2. Obertura i tancament de fitxers

En aquest apartat veurem les funcions principals que aporta el llenguatge C per a obertura i tancament: `fopen`, `fclose`, `fcloseall` i `freopen`. Treballarem, també, les funcions `fflush` i `flushall`, que tot i no correspondre a l'obertura i el tancament de fitxers hi estan relacionades.

1) Funció `fopen`

La funció del llenguatge C que permet obrir un fitxer és aquesta:

```
#include <stdio.h>
FILE *fopen (const char *nomFitxerExtern, const char *mode)
```

L'argument `nomFitxerExtern` és una cadena que conté el nom del fitxer extern amb el camí (en cas contrari al SGF el cerca en el directori

actual). L'argument `mode` es refereix al tipus d'obertura del fitxer. La taula 8 ens mostra les diferents possibilitats per a l'argument `mode`.

Taula 8. Possibilitats per a l'argument `mode` en la funció `fopen` del llenguatge C

Mode	Funcionalitat
`"r"`	Accés per a lectura; si el fitxer no existeix o no es troba, retorna un error.
`"w"`	Accés per a escriptura; si el fitxer no existeix, es crea, i si existeix, el seu contingut es destrueix i és creat de nou sense guardar-ne còpia de seguretat; mentre està obert és permès escriure al final (afegint) o damunt la informació existent.
`"a"`	Accés per a afegir informació al final; si el fitxer no existeix, es crea.
`"r+"`	Accés per a llegir i escriure amb la funcionalitat del mode w; el fitxer ha d'existir.
`"w+"`	Accés per a llegir i escriure; si el fitxer no existeix, es crea, i si existeix, el seu contingut es destrueix i és creat de nou sense guardar-ne còpia de seguretat.
`"a+"`	Accés per a llegir i afegir; si el fitxer no existeix, es crea.

Ha de quedar molt clar que el mode `"a"` només permet afegir pel final del fitxer; no es deixa escriure damunt d'informació ja existent.

Si l'execució de la funció `fopen` té èxit, retorna un punter a `FILE`, el qual és l'equivalent del fitxer intern. En totes les operacions posteriors, amb el fitxer es farà referència a aquest punter. Per contra, si l'obertura no es pot realitzar, la funció retorna el punter `NULL`.

Però les possibilitats d'obertura no acaben en aquests sis modes. Cadascun pot anar seguit d'una `b` (binari) o una `t` (text). Vegem què impliquen l'una i l'altra.

De tots és conegut que el llenguatge C utilitza com a final de línia el caràcter `'\n'`, és a dir, el caràcter 10 del codi ASCII (Line Feed). D'altra banda, al final d'un fitxer, el llenguatge C no situa cap marca. Aquest funcionament del llenguatge C és el que fa servir el sistema operatiu Unix-Linux amb els seus fitxers. Però el sistema operatiu DOS-Windows té un funcionament diferent, ja que marca el final de línia amb dos caràcters (Carriage Return - ASCII 13 - i Line Feed - ASCII 10 -) i el final de fitxer amb un caràcter (End Of File - ASCII 26 -). Com que el funcionament dels dos sistemes operatius és diferent, implementacions del llenguatge C funcionen diferent, de manera que un programa C sobre DOS-Windows converteix `'\n'` en CR+LF en una instrucció d'escriptura a fitxer i converteix la parella CR+LF en `'\n'` en una instrucció de lectura de fitxer. El mateix programa C sobre Unix-Linux no faria aquestes transformacions. Aquesta conversió pot ocasionar problemes quan aquests caràcters no corresponguin a un text sinó que corresponguin a un valor numèric que s'està recuperant d'un fitxer de disc i quan dos bytes consecutius corresponguin a CR+LF. Respecte al caràcter EOF, el llenguatge C no l'escriu mai (ni en DOS-Windows ni en

Unix-Linux). Per a desactivar la transformació `'\n'` - CR+LF cal utilitzar el mode binari. Aquesta opció en Unix-Linux és ignorada encara que sintàcticament és acceptada ja que permet la portabilitat d'un programa entre els sistemes operatius. Si no s'especifica mode text ni mode binari, s'adopta el contingut de la variable global _fmode que per defecte té el valor text.

2) Funció `fclose`

La funció del llenguatge C que permet tancar un fitxer és aquesta:

```
#include <stdio.h>
int fclose (FILE *f)
```

Aquesta funció retorna zero si el tancament ha estat satisfactori i un valor `EOF` en cas d'error.

3) Funció `fcloseall`

El llenguatge C aporta la possibilitat de tancar tots els streams oberts mitjançant la utilització d'una única funció.

```
#include <stdio.h>
int fcloseall (void);
```

Aquesta funció retorna el nombre total de fitxers tancats si no hi ha hagut cap error i un valor `EOF` en cas d'error. És interessant utilitzar aquesta funció quan no es té necessitat de controlar el bon tancament de cap fitxer. En cas contrari, caldrà fer servir la funció `fclose` per a tancar els fitxers a controlar i, posteriorment, utilitzar la funció `fcloseall` per a tancar la resta.

4) Funció `freopen`

El llenguatge C permet, amb una única instrucció, tancar un fitxer obert i utilitzar el mateix stream per a obrir-ne un altre. La sintaxi corresponent és la següent:

```
#include <stdio.h>
FILE *freopen (const char *nomFitxerExtern, const char *mode, FILE *fitxer)
```

L'argument `fitxer` fa referència al fitxer intern obert que cal tancar i tornar a obrir vinculant-lo al fitxer extern `nomFitxerExtern`.

Aquesta funció és utilitzada per a redireccionar els streams predefinits `stdin`, `stdout` o `stderr` a fitxers especificats per l'usuari.

5) Funcions `fflush` i `flushall`

Ja sabem l'existència de la funció `fflush` per a buidar els buffers associats als streams d'E/S per teclat i pantalla. Aquesta funció és utilitzable en qualsevol stream i, per tant, es pot fer servir per a buidar els buffers d'E/S en fitxers. Recordem-ne la sintaxi:

```
#include <stdio.h>
int fflush (FILE *f);
```

Aquesta funció buida el buffer associat a l'stream `f` i retorna `EOF` si ha tingut lloc algun error o zero si l'operació s'ha executat correctament. L'stream resta obert després de l'execució de la funció `fflush`.

La funció `fflush` en gravació en fitxers és especialment útil, ja que fent-la servir es pot obligar que el SGF enregistri el que hi hagi en el buffer i no esperi una gravació més tardana. La seva utilització facilita la no comprovació d'error en l'execució de les funcions `fclose` i `fcloseall`.

De la mateixa manera que existeix la funció `fcloseall` associada a la funció `fclose`, per a la funció `fflush` també existeix la corresponent `flushall` en plataformes Turbo C però no pas en plataformes gcc.

```
#include <stdio.h>
int flushall (void);
```

Aquesta funció neteja els buffers per tots els streams d'entrada i provoca les sortides cap al fitxer dels continguts pendents en els buffers de sortida. Retorna el nombre d'streams oberts i tancats. En plataformes gcc, per aconseguir l'efecte de la funció `flushall`, cal utilitzar la funció `fflush` passant el punter `NULL` com a paràmetre.

2.3. Detecció d'errors comunicats pel SGF

En les operacions amb fitxers és important disposar d'eines que permetin detectar els errors que es puguin anar produint. El SGF del llenguatge C no avorta l'execució del programa. És responsabilitat del programador controlar els errors.

Recordem que el tractament d'error en pseudocodi depèn del valor que retornen les diferents funcions (un valor zero indica que l'operació ha estat executada correctament). En el llenguatge C no és la pròpia funció el que ens informa de l'estat d'execució. S'utilitza un altre sistema.

Cada fitxer obert té un indicador d'error (dins l'estructura FILE corresponent) que s'activa en produir-se un error i que es manté activat fins que el fitxer es tanca o es desactiva explícitament. Caldrà consultar l'indicador d'error (directament mitjançant els corresponents flags de l'estructura FILE –camí complicat– o amb la funció `ferror` –camí fàcil–) per a saber l'estat d'execució de les operacions. És molt important anar controlant els possibles errors a cada operació amb el fitxer, ja que, si no, l'indicador d'error activat no tindria per què correspondre a la darrera operació efectuada amb el fitxer.

A més de l'indicador d'error, cada fitxer obert manté també un indicador de final de fitxer que s'activa quan s'intenta accedir més enllà del darrer element. També caldrà consultar aquest indicador (per via directa –camí complicat– o mitjançant la funció `feof` –camí fàcil–).

Tingueu present que aquests dos indicadors es poden emprar per a fitxers oberts. Per tant, no són utilitzables en les operacions d'obertura i tancament. En aquests casos cal fer servir els valors que retornen les pròpies funcions.

1) Funció `ferror`

```
#include <stdio.h>
int ferror (FILE *f);
```

Aquesta funció efectua la consulta de l'indicador d'error del fitxer `f` i retorna zero si no hi ha hagut cap error i un valor diferent de zero en cas contrari.

Cal posar especial atenció en el funcionament d'aquesta funció doncs no és idèntic en totes les plataformes. Així, l'error provocat per l'execució de la funció `fprintf` (similar a `printf` i que permet efectuar una sortida formatada cap un fitxer) sobre un fitxer obert en mode només lectura, no és detectat per la funció `ferror` en els compiladors gcc mentre sí ho és en altres plataformes (com Turbo C).

2) Funció `clearerr`

```
#include <stdio.h>
void clearerr (FILE *f);
```

Aquesta funció desactiva l'indicador d'error i l'indicador de final de fitxer per al fitxer f i els posa a zero.

3) Funció `feof`

En la introducció del SGF seqüencial genèric hem comentat que després de cada instrucció de lectura cal comprovar si veritablement s'ha efectuat una lectura o si per contra s'ha arribat al final del fitxer, cosa que es pot detectar gràcies al valor que retorna la instrucció de lectura.

En el llenguatge C el funcionament és similar. Quan una operació de lectura sobre un fitxer intenta llegir més enllà de la marca de final de fitxer, el SGF activa automàticament l'indicador de final de fitxer corresponent.

La sintaxi de la funció `feof` és aquesta:

```
#include <stdio.h>
int feof (FILE *f);
```

Aquesta funció efectua la consulta de l'indicador de final de fitxer i retorna un valor diferent de zero quan s'ha arribat al final de fitxer i un zero en cas contrari.

2.4. Gestió d'errors a nivell de sistema operatiu

En ocasions es produeixen errors que són detectats pel sistema operatiu i ens pot ser d'interès conèixer com podem assabentar-nos-en en els nostres programes (no només quan es treballa amb fitxers). Per altra banda, davant certs errors excepcionals ens pot ser necessari avortar l'execució del programa sense arribar a la seva normal finalització. El llenguatge C ens proporciona dues funcions per aquestes situacions.

1) Funció `perror`

Els indicadors d'error i de final de fitxer són gestionats pel SGF que incorpora el llenguatge C. A vegades hi ha errors d'execució (no solament en el tractament de fitxers) que són detectats pel mateix sistema operatiu. La funció `perror` facilita la visualització de l'error que ha detectat el sistema operatiu corresponent. Per al cas del DOS són bastant limitats o inexistents.

La sintaxi de la funció `perror` és aquesta:

```
#include <stdio.h>
void perror (const char *missatge);
```

Aquesta funció escriu a la sortida estàndard `stderr` el `missatge` especificat en l'argument seguit de dos punts, més el missatge d'error del sistema operatiu i d'un Line Feed.

Per entendre com funciona la funció `perror`, primer cal conèixer l'existència de les tres variables globals següents:

```
int errno;                          /* emmagatzema el número d'error */
const char *const sys_errlist[];    /* emmagatzema els missatges d'error */
int sys_nerr;                       /* grandària de sys_errlist */
```

Aquestes variables poden residir en diferents fitxers de capçalera segons la plataforma. Així, per exemple, en Turbo C resideixen dins `stdlib.h` i en plataformes gcc dins `errno.h` i `stdio.h`.

Quan el sistema operatiu detecta un error, aquest queda enregistrat a la variable `errno`. La funció `perror` simplement comprova el valor de la variable `errno` (que està emmagatzemant el darrer error produït) i accedeix a la posició `errno` de la taula `sys_errlist`, on troba el missatge corresponent. En iniciar l'execució d'un programa, la variable `errno` val zero, valor que indica que no s'ha produït cap error. Atenció! És responsabilitat del programador netejar la variable `errno` (posar-la a zero) doncs sempre conté el valor del darrer error de sistema produït, el qual es pot haver produït força estona abans de la instrucció a controlar.

Anem a dissenyar un programa per a veure els errors que és capaç de reproduir la funció `perror`. Per aconseguir-ho no hem de fer altra cosa que llistar el contingut de la taula `sys_errlist`.

Trobareu el fitxer `u5n2p01.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* u5n2p01.c */

#include <stdio.h>
#include <conio.h>

extern const char *const sys_errlist[];
extern int sys_nerr;

void main(void)
{
   int i;
   clrscr();
   printf ("La taula sys_errlist conté %d missatges",sys_nerr);
   printf ("\nPremi qualsevol tecla per a visualitzar-los...");
   getch();
   clrscr();
   for (i=0; i<sys_nerr;i++)
   {
      if (i && i%24==0) { printf ("Premi tecla per continuar...\n"); getch(); clrscr();}
      printf ("Error: %d, %s\n",i,sys_errlist[i]);
   }
   printf ("Premi tecla per finalitzar...\n"); getch(); clrscr();}
}
```

La seva execució en diferents plataformes ens demostra que la quantitat de missatges a gestionar és diferent. Així, en Turbo C hi ha 36 missatges mentre que en gcc n'hi ha 126.

2) Funció `exit`

El llenguatge C proporciona, com la immensa majoria dels llenguatges, instruccions per a avortar l'execució sense haver arribat al final del flux del codi. Utilitzar aquestes funcions va, en principi, en contra de la programació estructurada i modular, que defensa que el flux del programa s'ha de seguir fins a la seva fi.

A vegades, però, en els programes es produeixen situacions excepcionals d'error que reclamen l'avortament de l'execució. Aconseguir que en aquestes situacions el flux del programa continuï fins a la fi del codi és complicat i empitjora la llegibilitat del programa. Per això és lícit, en programació estructurada i modular, utilitzar funcions que permetin l'avortament de l'execució davant situacions excepcionals.

La funció `exit` és un dels mecanismes que facilita el llenguatge C. La seva sintaxi és:

```
#include <stdlib.h>
void exit (int estat);
```

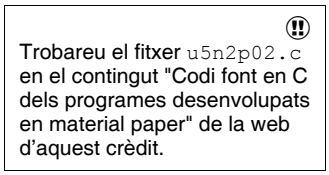
Aquesta funció finalitza l'execució del programa i retorna al sistema operatiu el valor enter `estat`. Abans d'acabar, els buffers de sortida són buidats cap als dispositius corresponents i els fitxers oberts són tancats. Com que els programes que acaben correctament acostumen a retornar un valor zero, quan utilitzem la funció `exit` per avortar el programa davant situacions excepcionals, serà lògic emprar valors diferents de zero, els quals seran detectables des del sistema operatiu per actuar, en funció d'aquests, d'una manera o d'una altra.

Vegem ara un programa en el que utilitzem les funcions `ferror`, `perror` i `exit`, per a veure'n el seu funcionament davant els diversos errors que es produiran. Així, en el programa provocarem diverses situacions: intentarem obrir un fitxer inexistent, crearem un fitxer, obrirem un fitxer en només lectura, hi intentarem escriure, tancarem els fitxers... Després de cada situació susceptible d'error visualitzarem el contingut de la variable `errno` i invocarem la funció `perror`.

Emprarem en aquest programa la funció `fprintf` de sortida amb format, per a la que no en coneixem encara la utilització en fitxers, però que és molt similar a la funció `printf` (té igual sintaxi amb un primer argument de tipus `FILE`) :

```
int fprintf (variable_FILE, "...", ...);
```

Aquesta funció, a l'igual que la funció `printf`, retorna el número de valors visualitzats. En cas d'error d'execució (per exemple quan el fitxer és obert en mode només lectura, retorna -1). En plataformes Turbo C, la funció `ferror` detecta la situació, mentre que en plataformes gcc, no se n'assabenta.


Trobareu el fitxer `u5n2p02.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* u5n2p02.c */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <errno.h>

void main()
{
   FILE *f;
   int i;
   clrscr();
   if ((f = fopen ("FITXER.TXT","r")) == NULL)
   {
     printf ("No s'ha pogut obrir el fitxer de nom FITXER.TXT.\n");
     if (errno)
     {  printf("errno=%d\n",errno); perror ("perror"); errno=0;}
     else printf("L'error no ha arribat al S.O.\n");
     if ((f = fopen ("FITXER.TXT","w")) == NULL)
     {
        printf ("\nNo s'ha pogut crear el fitxer de nom FITXER.TXT.\n");
        if (errno)
        {  printf("errno=%d\n",errno);perror ("perror"); errno=0;}
        else printf("L'error no ha arribat al S.O.\n");
        exit (1);
     }
     printf ("\nS'ha creat el fitxer de nom FITXER.TXT.\n");
     if (fclose (f))
     {
        printf ("\nEl fitxer de nom FITXER.TXT no s'ha tancat correctament.\n");
        if (errno)
        {  printf("errno=%d\n",errno);perror ("perror"); errno=0;}
        else printf("L'error no ha arribat al S.O.\n");
        exit (1);
     }
     printf ("\nS'ha tancat el fitxer de nom FITXER.TXT.\n");
     if ((f = fopen ("FITXER.TXT","r")) == NULL)
     {
        printf ("\nNo s'ha pogut obrir el fitxer de nom FITXER.TXT.\n");
        if (errno)
        {  printf("errno=%d\n",errno);perror ("perror"); errno=0;}
        else printf("L'error no ha arribat al S.O.\n");
        exit (1);
     }
   }
   printf ("\nS'ha obert el fitxer de nom FITXER.TXT en lectura.\n");
   printf ("\nIntentem escriure-hi. No ha de poder ser!\n");
   i=fprintf (f,"Això és una prova.\n");
```

```
   printf ("La funció fprintf retorna %d\n",i);
   if (ferror(f))
   {
      clearerr(f);
      printf ("\nError detectat per ferror en intentar escriure.\n");
      if (errno)
      {  printf("errno=%d\n",errno);perror ("perror"); errno=0;}
      else printf("L'error no ha arribat al S.O.\n");
   }
   if (fclose (f))
   {
      printf ("\nEl fitxer de nom FITXER.TXT no s'ha tancat correctament.\n");
      if (errno)
      {  printf("errno=%d\n",errno);perror ("perror"); errno=0;}
   }
}
```

Suposem que no existeix un fitxer de nom `FITXER.TXT`. Obtenim:

```
No s'ha pogut obrir el fitxer de nom FITXER.TXT.
errno=2
perror: No such file or directory

S'ha creat el fitxer de nom FITXER.TXT.

S'ha tancat el fitxer de nom FITXER.TXT.

S'ha obert el fitxer de nom FITXER.TXT en lectura.

Intentem escriure-hi. No ha de poder ser!
La funció fprintf retorna -1
Error detectat per ferror en intentar escriure.
errno=2
perror: No such file or directory
```

Les tres darreres línies apareixen en Turbo C però no pas en gcc. Això és motivat per que en gcc, la funció `ferror` no s'assabenta de l'error que es produeix quan s'intenta escriure amb la funció `fprintf` en un fitxer obert en mode de només lectura. En canvi, si haguéssim utilitzat la funció d'escriptura en fitxers `fwrite` (que veurem més endavant), la funció `ferror` hagués informat de l'error.

En conclusió, cal tenir molt clar els diferents comportaments per a dissenyar programes que controlin les diferents possibilitats d'error.

2.5. Gestió de directoris segons el sistema operatiu

Com es pot gestionar des de programa el contingut dels directoris? Com es pot saber si existeix un fitxer en un directori sense haver de procedir a una instrucció d'obertura? La resposta és diferent segons el sistema operatiu.

1) En implementacions de C sobre DOS (com Turbo C)

Les implementacions de C sobre DOS incorporen les funcions específiques següents:

```
#include <dir.h>
int findfirst (const char *nom_fitxer, struct ffblk *ffblk, int atribut);
int findnext (struct ffblk *ffblk);

struct ffblk
{
   char reservat[21];
   char atribut;
   unsigned hora, data;
   long tamany;
   char nom[13];
}
```

La funció `findfirst` permet iniciar la recerca de fitxers en un directori i la funció `findnext` permet continuar-la. Totes dues funcions retornen l'enter zero si s'ha trobat un fitxer i l'enter diferent de zero en cas contrari.

L'argument `nom_fitxer` ha de contenir el nom del fitxer que s'ha de cercar, el qual pot contenir els caràcters comodí * i ?. L'estructura `ffblk` emmagatzema la informació del fitxer trobat (si és el cas) i és utilitzable pel programa i per la funció `findnext` en cas que es vulgui continuar la recerca. Consulteu el manual de referència del llenguatge per a tenir més informació referent al contingut de l'estructura `ffblk`. L'argument `atribut` fa referència al tipus de fitxers que es vol cercar i cal omplir-lo amb una de les constants simbòliques (declarades a `dos.h`) següents:

`FA_RDONLY`	Fitxer només de lectura
`FA_HIDDEN`	Fitxer amagat
`FA_SYSTEM`	Fitxer de sistema
`FA_LABEL`	Etiqueta de volum
`FA_DIREC`	Directori
`FA_ARCH`	Fitxer

El següent programa efectua un llistat dels fitxers continguts en el directori actiu (només en plataformes DOS).

```
/* u5n2p03.c */

#include <stdio.h>
#include <dir.h>
#include <conio.h>
```

Trobareu el fitxer `u5n2p03.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
void main(void)
{
   struct ffblk info;
   int final;
   clrscr();
   printf("Llistat del directori actiu:\n");
   final = findfirst("*.*",&info,0);
   while (!final)
   {
      printf("  %s\n", info.ff_name);
      final = findnext(&info);
   }
}
```

Altres funcions interessants per a la gestió de directoris en plataformes DOS, són:

```
#include <dir.h>
int chdir (const char *path);
```

Permet canviar el directori actiu, el qual cal indicar en l'argument `path`. Retorna el valor zero si el canvi ha tingut lloc.

Aquesta funció també existeix en plataformes Linux, definida al fitxer de capçalera `unistd.h`.

```
include <dir.h>
char *getcwd (char *buf, int buflen);
```

Permet recollir el camí sencer del directori actiu (inclòs el nom del disc) fins la llargada màxima indicada a `buflen` i el deixa en la variable `buf`. Si la llargada del camí és superior al valor de `buflen` es produeix un error. Si el punter `buf` indicat val `NULL`, llavors la plataforma crea dinàmicament una zona de memòria per encabir-hi el camí sencer. La funció retorna `NULL` si hi ha algun tipus d'error o la direcció de `buf` o de la zona de memòria creada plena amb el contingut del camí sencer.

Aquesta funció també existeix en plataformes Linux, definida al fitxer de capçalera `unistd.h`.

```
include <dir.h>
int getdisk(void);
```

Permet recollir el disc actiu. Retorna el valor 0 per al disc `A`, 1 per al disc `B`, etcètera.

```
include <dir.h>
int setdisk (int drive);
```

Intenta fixar el disc actiu a l'indicat per `drive`, tenint en compte que el valor zero és per al disc A, 1 per al disc B, etcètera. Si no existeix el `drive` indicat, no efectua cap canvi. Retorna el nombre de discos accessibles.

```
include <dir.h>
int mkdir (const char *path);
```

Intenta crear un nou directori indicat per `path`. Retorna zero si s'ha pogut crear el nou directori. En cas d'error, amb la variable `errno` se'n pot conèixer el motiu.

```
include <dir.h>
int rmdir (int drive);
```

Intenta eliminar el directori indicat per `path`, el qual ha d'estar buit, no pot ser el directori actiu i tampoc pot ser el directori arrel. Retorna zero si s'ha pogut eliminar el directori indicat. En cas d'error, amb la variable `errno` se'n pot conèixer el motiu.

2) En implementacions amb gcc

Principalment cal tenir en compte les següents funcions:

```
#include <dirent.h>
DIR *opendir (const char *nom);
struct dirent *readdir (DIR *directori_stream_obert);
void rewinddir (DIR *directori_stream_obert);
off_t telldir (DIR *directori_stream_obert);
void seekdir (DIR *directori_stream_obert, off_t offset);
int *closedir (DIR *directori_stream_obert);

struct dirent
{
   ...
   char d_name[];
   ...
}
```

La funció `opendir` intenta obrir un flux (`stream`) al directori indicat pel paràmetre `nom`. En cas d'èxit, retorna el punter a l'`stream` apuntat (`DIR*`) i en cas d'error retorna `NULL`. La variable `errno` permet deduir la causa d'error.

Una vegada tenim un directori obert (apuntat per un `DIR*`), la funció `readdir` ens permet fer un recorregut pel contingut del directori. A cada execució de la funció `readdir`, aquesta intenta situar-se a la següent entrada del directori, retornant un `struct dirent *` si ha pogut situar-se en una entrada o `NULL` quan ja no hi ha més entrades. El

punter `DIR *` guarda la informació de l'entrada apuntada a cada moment. L'atribut `d_name` de l'estructura `dirent` dona informació referent al nom de l'objecte existent a l'entrada.

La funció `rewinddir` permet resituar el punter DIR * a l'inici del directori, de manera que una execució de la funció `readdir` facilitaria la primera entrada del directori.

La funció `telldir` permet conèixer en quina posició del directori obert (apuntat per l'argument `DIR*`) estem situats.

La funció `seekdir` permet situar-se en una posició determinada del directori obert (apuntat per l'argument `DIR*`). El segon argument (posició) ha d'haver estat obtingut per la funció `telldir`.
La funció `closedir` tanca el directori obert (apuntat per un DIR *). Una vegada el directori és tancat, no és accessible per cap de les funcions `readdir`, `rewinddir`, `telldir` i `seekdir`.

El següent programa efectua un llistat dels fitxers continguts en el directori actiu (només en plataformes gcc).

Trobareu el fitxer `u5n2p04.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* u5n2p04.c */

#include <stdio.h>
#include <dirent.h>
#include <conio.h>
#include <stdlib.h>
#include <errno.h>

#define MAX_LEN_PATH 255

void main(void)
{
   char nom[MAX_LEN_PATH];
   struct dirent *info;
   DIR *dir;
   clrscr();
   getcwd(nom,MAX_LEN_PATH);
   if (errno==ERANGE)
   {
      printf("El camí del directori actiu necessita més de MAX_LEN_PATH caràcters.");
      exit(1);
   }
   printf("Nom del directori actiu: %s\n",nom);
   printf("Llistat del directori actiu:\n");
   dir = opendir(nom);
   if (!dir)
   {
      printf("No es pot obrir el directori actiu.");
      exit(1);
   }
   while ((info=readdir(dir))!=NULL)
```

```
   {
      printf("  %s\n", info->d_name);
   }
}
```

3) Nova versió dels fitxers `verscomp.h` i `verscomp.c` per continuar mantenint compatibilitat en els nostres exercicis per a les plataformes Turbo C i gcc.

Us atreviu a dissenyar la funció següent per a la plataforma Turbo C i per a la plataforma gcc?

```
int existeix_fitxer(const char *nom);
/* Comprova si existeix el fitxer de nom indicat en l'argument.
   Retorna: 0: Hi és
            -1: No hi és
*/
```

Ens interessa afegir aquesta nova funcionalitat dins els fitxers `verscomp.h` i `verscomp.c` desenvolupats fins el moment per tal de mantenir la compatibilitat dels nostres exemples i exercicis per les dues plataformes.

Trobareu la nova versió dels fitxers verscomp-`t.h` i verscomp-t.c per a Turbo C i dels fitxers verscomp-g.h i verscomp-g.c per a compiladors gcc, en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* Per a Turbo C */

#include <stdio.h>
#include <dir.h>
#include <dos.h>

int existeix_fitxer(const char *nom)
{
   struct ffblk info;
   return findfirst(nom, &info, FA_ARCH);
}
```

```
/* Per a compiladors gcc */

#include <unistd.h>
#include <errno.h>
#include <dirent.h>

#define MAX_LEN_PATH 1000

int existeix_fitxer(const char *nom)
{
   char dirActual[MAX_LEN_PATH];
   DIR *dir;
   struct dirent *info;

   getcwd(dirActual,MAX_LEN_PATH);
   if (errno==ERANGE) return -1;

   dir = opendir (dirActual);
   if (!dir) return -1;
```

```
   info = readdir (dir);
   while (info && strcmp(info->d_name,nom)!=0)
      info = readdir (dir);

   closedir(dir);
   if (!info) return -1;

   return 0;
}
```

2.6. E/S caràcter a caràcter

Les dades en fitxers poden ser escrites i llegides caràcter a caràcter mitjançant les funcions que presentem a continuació.

1) Funció `fputc`

La seva sintaxi és aquesta:

```
#include <stdio.h>
int fputc (int caràcter, FILE *f);
```

Aquesta funció envia el caràcter a l'stream `f`. Si l'enviament ha estat correcte retorna el caràcter enviat. En cas contrari, retorna el caràcter `EOF`. Ara bé, com que `EOF` és pròpiament un caràcter, caldrà utilitzar la funció `ferror` per a detectar si s'ha produït un error quan el caràcter a enviar sigui el mateix `EOF`.

2) Funció `fgetc`

La seva sintaxi és aquesta:

```
#include <stdio.h>
int fgetc (FILE *f);
```

Aquesta funció llegeix un caràcter de l'stream `f`, el qual és el valor retornat. En cas que no es pugui llegir cap caràcter (per un error de lectura o perquè s'hagi arribat al final del fitxer) retorna el valor `EOF`. Ara bé, igual que en la funció `fputc`, el valor `EOF` és un valor acceptat i, per tant, caldrà utilitzar les funcions `ferror` i `feof` per a detectar, respectivament, si s'ha produït un error o si s'ha arribat al final del fitxer.

3) Exemple d'utilització de les funcions `fputc` i `fgetc`

Ara farem un programa que demani un text a l'usuari pel teclat i posteriorment l'enregistri, caràcter a caràcter, en un fitxer de nom `FITXER.TXT`, el qual haurà de ser creat pel programa. Posteriorment, el mateix programa procedirà a recuperar el text del fitxer i a mostrar-lo per pantalla.

Trobareu el fitxer `u5n2p05.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* u5n2p05.c */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

void main()
{
   char c;
   FILE *f;
   clrscr();
   if ((f = fopen ("FITXER.TXT","w")) == NULL)
   {
     fprintf (stderr, "Error en crear el fitxer FITXER.TXT. Avortem el programa.\n");
     exit (1);
   }
   printf ("Introdueixi un text.\n");
   printf ("L'aparició del símbol $ indicarà la fi.\n\n");
   scanf("%c",&c);
   while (c != '$')
   {
      fputc (c,f);
      if (ferror(f))
      {
        fprintf (stderr, "Error en gravar un caràcter al fitxer FITXER.TXT.\n");
        clearerr (f);
      }
      scanf("%c",&c);
   }
   printf ("\n\nS'ha finalitzat la lectura de teclat i gravació en fitxer.\n");
   if (fclose(f))
   {
     fprintf (stderr, "Error en tancar el fitxer FITXER.TXT. Avortem el programa.\n");
     exit (1);
   }
   printf ("\nPremi una tecla per procedir a la lectura del fitxer.\n");
   getch();
   clrscr();
   if ((f = fopen ("FITXER.TXT","r")) == NULL)
   {
     fprintf (stderr, "Error en obrir el fitxer FITXER.TXT. Avortem el programa.\n");
     exit (1);
   }
   printf ("El text llegit del fitxer és:\n\n");
   c = fgetc (f);
   while (!ferror(f) && !feof(f))
   {
      printf ("%c",c);
```

```
      c = fgetc (f);
   }
   if (ferror(f))
   {
     fprintf (stderr, "Error en llegir del fitxer FITXER.TXT. Avortem el programa.\n");
     exit (1);
   }
   printf ("\n\nTot el text del fitxer ha estat llegit.\n");
   fclose (f);
}
```

Observem que després de cada operació amb fitxer comprovem que sigui correcte. Hi ha una excepció, però. En el darrer tancament del fitxer no fem cap comprovació. Això és així perquè en aquest cas el fitxer era obert en lectura i, per tant, en tancar el fitxer no hi pot haver coses pendents de gravació. En canvi, en el primer tancament del fitxer, quan aquest era obert en escriptura i s'hi havien gravat coses, sí que comprovem la validesa del tancament, ja que en aquests moments hi pot haver caràcters que són traspassats al fitxer des del buffer.

Observem també que a cada lectura del fitxer es comprova l'indicador d'error i l'indicador de final de fitxer, de manera que l'activació de qualsevol dels dos finalitza el procés de lectura. Evidentment, cal esbrinar el motiu que ha fet acabar el procés, cosa que fem preguntant si s'ha produït error. Si no s'ha produït error és que, forçosament, s'ha arribat al final del fitxer.

2.7. E/S enter a enter

Les dades en fitxers poden ser escrites i llegides enter a enter (en mode binari) mitjançant les funcions que presentem a continuació.

1) Funció `putw`

La seva sintaxi és aquesta:

```
#include <stdio.h>
int putw (int i, FILE *f);
```

Aquesta funció envia la dada binària de tipus enter `i` a l'stream `f`. Si l'enviament ha estat correcte retorna el valor enviat. En cas contrari, retorna el valor `EOF`. Ara bé, com que `EOF` és pròpiament un enter, caldrà utilitzar la funció `ferror` per a detectar si s'ha produït un error quan el valor que s'ha d'enviar sigui el mateix `EOF`.

Una altra nomenclatura

En certes publicacions les E/S enter a enter s'anomenen E/S paraula a paraula, amb el benentès que es parla de paraula màquina, la qual és una dada del tipus `int`. Així es pot interpretar el sufix `w` per a finalitzar el nom de les funcions, ja que correspon a la paraula `word`.

2) Funció `getw`

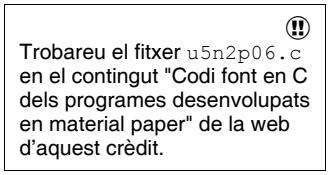
La seva sintaxi és aquesta:

```
#include <stdio.h>
int getw (FILE *f);
```

Aquesta funció llegeix una dada entera binària de l'stream `f`, el qual és el valor retornat. En cas que no es pugui llegir cap valor (per un error de lectura o perquè s'hagi arribat al final del fitxer) retorna el valor `EOF`. Ara bé, igual que en la funció `putw`, el valor `EOF` és un valor acceptat i, per tant, caldrà utilitzar les funcions `ferror` i `feof` per a detectar, respectivament, si s'ha produït un error o si s'ha arribat al final del fitxer.

3) Exemple d'utilització de les funcions `putw` i `getw`

Ara farem un programa que emmagatzemi els cent primers valors enters en forma binària en un fitxer de nom `FITXER.TXT`, el qual haurà de ser creat pel programa. Posteriorment, el mateix programa procedirà a recuperar els valors del fitxer i a mostrar-los per pantalla.




```
/* u5n2p06.c */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

void main()
{
   unsigned int i;
   FILE *f;
   clrscr();
   if ((f = fopen ("FITXER.TXT","wb")) == NULL)
   {
     fprintf (stderr, "Error en crear el fitxer FITXER.TXT. Avortem el programa.\n");
     exit (1);
   }
   for (i=0;i<100;i++)
   {
      putw(i,f);
      if (ferror(f))
      {
       fprintf (stderr,"\n\nS'ha produït error en gravar l'enter %3d al fitxer.\n",i);
       clearerr(f);
      }
   }
   printf ("\n\nS'ha finalitzat la gravació en fitxer.\n");
   if (fclose(f))
   {
     fprintf (stderr, "Error en tancar el fitxer FITXER.TXT. Avortem el programa.\n");
     exit (1);
```

```
    }
    printf ("\nPremi una tecla per procedir a la lectura del fitxer.\n");
    getch();
    clrscr();
    if ((f = fopen ("FITXER.TXT","rb")) == NULL)
    {
      fprintf (stderr, "Error en obrir el fitxer FITXER.TXT. Avortem el programa.\n");
      exit (1);
    }
    printf ("Els valors llegits del fitxer són:\n\n");
    i = getw (f);
    while (!ferror(f) && !feof(f))
    {
       printf ("%2d ",i);
       if ((i+1)%20==0) printf("\n");
       i = getw (f);
    }
    if (ferror(f))
    {
      fprintf (stderr, "Error en llegir del fitxer FITXER.TXT. Avortem el programa.\n");
      exit (1);
    }
    printf ("\n\nTot el text del fitxer ha estat llegit.\n");
    fclose (f);
}
```

Seria bo que observéssiu (amb l'ordre `type` –DOS/Windows– o `cat` –Unix/Linux–) el contingut del fitxer `FITXER.TXT` enregistrat al disc. No veureu els valors numèrics, ja que han estat enregistrats com a valors binaris (2 o 4 bytes per cada `int`, depenent de la màquina). Observeu, també, la grandària del fitxer, que ha de ser de 200 octets, si un `int` ocupa 2 octets, i de 400 octets si n'ocupa 4. Finalment, observeu que hem hagut d'utilitzar el mode binari per a obrir el fitxer, tant en el moment d'escriptura com en el moment de lectura. Us proposo que executeu el programa canviant el mode binari pel mode text. Què succeeix? Per què?

2.8. E/S cadena a cadena

Les dades en fitxers poden ser escrites i llegides com a cadenes mitjançant les funcions que presentem a continuació.

1) Funció `fputs`

La seva sintaxi és aquesta:

```
#include <stdio.h>
int fputs (const char *cadena, FILE *f);
```

Aquesta funció envia la cadena a l'stream `f`. Si l'enviament ha estat correcte retorna el valor zero. En cas contrari, retorna un valor diferent de zero. El caràcter `'\0'` de final de cadena no s'envia a l'stream `f`.
Per a poder recuperar de forma senzilla la informació enregistrada al fitxer, és aconsellable, tal com es deduirà del funcionament de la funció `fgets`, enregistrar el caràcter `'\n'` després de cada cadena escrita.

2) Funció `fgets`

La seva sintaxi és aquesta:

```
#include <stdio.h>
char *fgets (char *cadena, int n, FILE *f);
```

Aquesta funció llegeix una cadena de l'stream `f` i l'emmagatzema a partir de la direcció de memòria apuntada per cadena. Es llegeixen caràcters des de la posició actual dins el fitxer fins al primer caràcter `'\n'` inclòs, fins al final del fitxer o fins que el nombre de caràcters llegit sigui igual a n-1. La cadena llegida queda acabada de manera automàtica amb el valor `'\0'`. El caràcter `'\n'` NO és eliminat en cas d'haver-lo llegit.

Enteneu ara el consell anterior referent a enregistrar el valor `'\n'` en finalitzar l'escriptura d'una cadena amb la funció `fputs`?

La funció `fgets` retorna un punter a l'inici de la cadena llegida. Si aquest punter és NULL indica que ha ocorregut un error o que s'ha arribat al final del fitxer. Cal utilitzar les funcions `ferror` i `feof` per a determinar la causa de la no lectura.

3) Exemple d'utilització de les funcions `fputs` i `fgets`

Ara farem un programa que demani un text a l'usuari pel teclat i posteriorment l'enregistri, línia a línia, en un fitxer de nom `FITXER.TXT`, el qual haurà de ser creat pel programa. Després, el mateix programa procedirà a recuperar el text del fitxer i a mostrar-lo per pantalla.

Definirem una cadena de capacitat màxima 80 caràcters per a emmagatzemar les línies que introdueixi l'usuari. Una línia de més grandària serà tractada en el programa a trossos, tot i que aconseguirem que en el fitxer i en la posterior visualització per pantalla es vegi com una línia sencera.

Trobareu el fitxer `u5n2p07.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* u5n2p07.c */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

#define MAX 80

void main()
{
   char s[MAX+1];
   FILE *f;
   clrscr();
   if ((f = fopen ("FITXER.TXT","w")) == NULL)
   {
     fprintf (stderr, "Error en crear el fitxer FITXER.TXT. Avortem el programa.\n");
     exit (1);
   }
   printf ("Introdueixi un text, separant les línies amb un <Entrar>.\n");
   printf ("Per finalitzar l'entrada introdueixi una línia buida.\n\n");
   while (fgets(s,MAX+1,stdin) && s[0]!='\n')
   {
      if (!fputs (s,f))
      {
        fprintf (stderr, "Error en gravar la cadena %s al fitxer FITXER.TXT.\n",s);
        clearerr (f);
      }
   }
   if (ferror(stdin))
   {
      fprintf (stderr, "Error en llegir del teclat. Avortem el programa.\n");
      exit (1);
   }
   printf ("\n\nS'ha finalitzat la lectura de teclat i gravació en fitxer.\n");
   if (fclose(f))
   {
     fprintf (stderr, "Error en tancar el fitxer FITXER.TXT. Avortem el programa.\n");
     exit (1);
   }
   printf ("\nPremi una tecla per procedir a la lectura del fitxer.\n");
   getch();
   clrscr();
   if ((f = fopen ("FITXER.TXT","r")) == NULL)
   {
     fprintf (stderr, "Error en obrir el fitxer FITXER.TXT. Avortem el programa.\n");
     exit (1);
   }
   printf ("El text llegit del fitxer és:\n\n");
   while (fgets(s,MAX+1,f)) fputs (s,stdout);
   if (ferror(f))
   {
     fprintf (stderr, "Error en llegir del fitxer FITXER.TXT. Avortem el programa.\n");
     exit (1);
   }
   printf ("\n\nTot el text del fitxer ha estat llegit.\n");
   fclose (f);
}
```

Observeu que, en aquest cas, hem aprofitat les funcions `fgets` i `fputs` per a llegir del teclat i per a escriure per pantalla. Hem pogut comprovar que les operacions amb fitxers són utilitzables en els diversos dispositius sempre que es treballi amb streams.

2.9. E/S amb format

Les dades en fitxers poden ser escrites i llegides com a cadenes mitjançant les funcions `fprintf` i `fscanf`, que presentem a continuació.

1) Funció `fprintf`

La seva sintaxi és aquesta:

```
#include <stdio.h>
int fprintf (FILE *f, const char *format [, arg]...);
```

Aquesta funció té el mateix funcionament que la coneguda funció `printf`. Si l'stream especificat en la funció `fprintf` és `stdout`, les funcions són totalment equivalents.

En utilitzar la funció d'escriptura amb format, és molt important conèixer com s'enregistren els valors numèrics. Recordeu com es visualitza un enter per pantalla quan es fa servir la funció `printf` amb el format `%d`? Es visualitzen els seus dígits, oi? És a dir, es requereix un octet per cada dígit. Aquesta no és la forma idònia d'emmagatzemar els valors numèrics, ja que s'ocupa molt espai en el disc. La manera idònia és emmagatzemar els valors numèrics ocupant els bytes necessaris (2, 4 o 8 en funció del tipus de dada). ‼

2) Funció `fscanf`

La seva sintaxi és aquesta:

```
#include <stdio.h>
int fscanf (FILE *f, const char *format [,arg]...);
```

Aquesta funció té el mateix funcionament que la coneguda funció `scanf`. Si l'stream especificat en la funció `fscanf` és `stdin`, les funcions són totalment equivalents.

2.10. E/S de blocs

En aquestes alçades ja coneixem funcions que ens permeten escriure i llegir en fitxers. Ara bé, recordeu quines són les funcions genèriques d'accés en un SGF? Les operacions de lectura i escriptura en un SGF sempre són considerades a escala de registre, és a dir, es llegeix un registre sencer, es modifica un registre sencer i s'escriu un registre sencer.

Sabem , també, que el SGF que aporta el llenguatge C és "especial". Ho és en el sentit que permet, tal com hem pogut comprovar, accedir a les dades mitjançant unitats d'accés molt inferiors al concepte de registre. Bé, si ens dóna aquesta funcionalitat, no la rebutjarem, oi? Però, i si volem treballar com un veritable SGF, és a dir, a escala de registre?

El llenguatge C aporta dues funcions per a escriure i llegir blocs de bytes (registres, si ho preferim) en fitxers.

1) Funció `fwrite`

La seva sintaxi és aquesta:

```
#include <stdio.h>
size_t fwrite (const void *ptr, size_t size, size_t n, FILE *f);
```

Aquesta funció envia `n` elements de grandària `size` octets emmagatzemats en la zona de memòria apuntada per `ptr` a l'stream `f` i retorna el nombre d'elements enviats a l'stream. Si aquest nombre és menor que `n`, és que hi ha hagut un error.
Aquesta funció emmagatzema les dades numèriques en format binari. És a dir, una dada del tipus `int` ocupa 2 o 4 octets (segons la màquina); una dada del tipus `long` n'ocupa 4 o 8; una dada del tipus `float` n'ocupa 4, i una dada del tipus `double` n'ocupa 8.

Distinció entre mode binari i format binari

No s'ha de confondre el mode binari d'obertura d'un fitxer que deshabilita la transformació de '\n' - CR+LF amb el format binari utilitzat per a emmagatzemar una dada numèrica.

2) Funció `fread`

La seva sintaxi és aquesta:

```
#include <stdio.h>
size_t fread (void *ptr, size_t size, size_t n, FILE *f);
```

Aquesta funció llegeix `n` elements de grandària `size` octets del dispositiu associat a l'stream `f`, els emmagatzema en la zona de memòria apuntada per `ptr` i retorna el nombre d'elements llegits. Si aquest nombre és menor que `n`, és que hi ha hagut un error. Cal utilitzar les funcions `feof` i `ferror` per a distingir si s'ha detectat el final de fitxer o

si ha tingut lloc un error. Si `size` o `n` són zero, la funció `fread` retorna un zero i el contingut de la zona de memòria apuntada per `ptr` no es modifica.

3. Gestió de fitxers seqüencials

En aquests moments coneixem els conceptes bàsics sobre SGF i sabem que els principals tipus de SGF que ens podem trobar són els seqüencials, els relatius i els seqüencial-indexats. Anem a estudiar en profundicat els SGF seqüencials, també anomenats "fitxers seqüencials".

Fixeu-vos que utilitzem la nomenclatura "*fitxer seqüencial*" quan en realitat "*seqüencial*" no és una característica del fitxer sinó del SGF que gestiona el fitxer.

Anem a veure, en primer lloc les operacions teòriques que ens acostumarem a trobar en un SGF seqüencial i posteriorment presentarem alguns algorismes bàsics de tractament seqüencial pels que plantejarem exemples en pseudocodi i en llenguatge C.

3.1. Operacions teòriques

Recordem la definició del concepte de fitxer seqüencial:

> Un fitxer seqüencial és aquell que permet l'accés seqüencial segons la posició física que ocupa dins el fitxer.

Per a aconseguir l'accés seqüencial, el SGF manté, per cada fitxer obert, un punter en el darrer registre llegit que sap avançar cap al registre següent quan s'efectua una nova lectura. Sap fer algun altre tipus de moviment? Sap fer salts? Sap tornar enrere? No, no i no. Un fitxer seqüencial només sap avançar cap endavant. Si sabés fer altres moviments ja no es tractaria d'un SGF seqüencial.

Sempre que es vol treballar amb un SGF (seqüencial o no) cal fer un estudi profund de les operacions que aporta per al seu funcionament. En aquest apartat presentarem, en pseudocodi, les operacions que trobarem en la majoria de SGF seqüencials. I, és clar, un fitxer ha de manipular-se amb les operacions que aporta el SGF corresponent.

Considerarem les operacions següents:

```
funció obrir_fs (f:fitxer_seq de T, F:cadena, mode:caràcter) retorna enter;
funció tancar_fs (f:fitxer_seq de T) retorna enter;
```

```
funció llegir_fs (f:fitxer_seq de T, r:T) retorna enter;
funció escriure_fs (f:fitxer_seq de T, r:T) retorna enter;
funció modificar_fs (f:fitxer_seq de T, r:T) retorna enter;
```

Observem, primer de tot, que totes les funcions tenen un argument `f` definit com a `fitxer_seq de T`. Aquest argument, associat a un fitxer extern, és el fitxer intern, amb el qual treballa el SGF i que està associat a un fitxer extern. Fixeu-vos que en definir el fitxer intern estem indicant el tipus seqüencial del fitxer i el tipus `T` dels seus registres. Evidentment, `T` ha de ser un tipus definit prèviament.

Cal advertir, també, que els noms de totes les funcions porten el sufix `fs` per a indicar que són operacions del SGF seqüencial. Més endavant veurem que les funcions dels SGF relatius portaran el sufix `fr` i les funcions dels SGF indexats, el sufix `fi`. D'aquesta manera estem distingint les funcions dels diferents SGF, les quals poden utilitzar-se simultàniament en un mateix programa.

Finalment, cal tenir en compte que totes les funcions retornen un enter. Aquest és el codi d'error per a conèixer si l'execució de la funció ha estat o no satisfactòria. Considerarem que una funció retornarà zero si l'execució ha estat correcta. En cas contrari, retornarà un valor que ens indicarà el tipus d'anomalia produïda.

1) Funció `obrir_fs`

```
funció obrir_fs (f:fitxer_seq de T, F:cadena, mode:caràcter) retorna enter;
```

La funció `obrir_fs` és la que efectua l'enllaç (i, per tant, reserva els canals i els buffers necessaris) entre el fitxer intern `f` (primer argument) i el fitxer extern `F` (segon argument). La cadena `F` ha de contenir el nom del fitxer extern i incorporar o no el camí per a trobar-lo. En cas de no incorporar el camí, al SGF cercarà el fitxer extern en el directori actual. El tercer argument `mode` permet definir el tipus d'obertura del fitxer; tenim quatre possibilitats: `I` (inici), `C` (consulta), `A` (actualització), `E` (extensió). Observem, en la taula 9, què es pot fer en funció del mode d'obertura del fitxer.

Taula 9. Tipus d'obertura d'un fitxer seqüencial.

Mode	Què està en disposició de. fer després d'obrir-se	Què es pot fer	Què no es pot fer
Inici	Escriure el primer registre	Escriure	Llegir, modificar
Consulta	Llegir el primer registre existent	Llegir	Escriure, modificar
Actualització	Llegir el primer registre existent	Llegir Modificar un registre existent	Escriure
Extensió	Escriure després del darrer registre	Escriure	Llegir, modificar

El mode inici implica la creació del fitxer extern, el qual podria o no existir abans. En cas que no existís, el crea. Però, i si ja existia? No tots els SGF tenen el mateix comportament davant d'aquesta situació. Trobem diferents casos:

- Sobreescriptura del fitxer existent, cosa que en provoca la pèrdua.
- Salvaguarda del fitxer existent amb un nom diferent, normalment afegint una extensió que indica la versió anterior. Només es guarda la versió anterior.
- Creació del fitxer amb control de versions, de manera que al SGF treballarà amb la darrera versió. Es van guardant totes les versions.

El mode consulta permet la lectura dels registres del fitxer. El fitxer extern ha d'existir.

El mode actualització permet la lectura dels registres i la seva modificació.

El mode extensió permet la inclusió de nous registres pel final del fitxer, és a dir, afegint registres als existents.

2) Funció `tancar_fs`

```
funció tancar_fs (f:fitxer_seq de T) retorna enter;
```

La funció `tancar_fs` desfà l'enllaç entre el fitxer intern i el fitxer extern (i, per tant, allibera els recursos assignats en l'obertura). Si el fitxer estava obert en els modes `inici`, `actualització` o `extensió` i en els buffers resten dades pendents de gravar en el fitxer extern (recordem que podia existir un funcionament diferit de l'enregistrament), es realitza la sortida física pertinent. Aquesta funció també retorna un codi d'error. Quin sentit té? Només té sentit quan hi pugui haver en el tancament algun enregistrament pendent.

3) Funció `llegir_fs`

```
funció llegir_fs (f:fitxer_seq de T, r:T) retorna enter;
```

La funció `llegir_fs` efectua la lectura lògica del registre següent des del buffer cap a la variable `r` (segon argument). Ja sabem que provocarà la lectura física si la informació cercada no es troba en el buffer. Després de l'execució d'aquesta funció s'ha de comprovar obligatòriament el valor retornat. Hi ha un valor (suposarem `FI_FITXER`) que ens fa saber que la lectura no ha pogut ser efectuada

ja que s'ha arribat a la fi del fitxer. És a dir, suposem que el nostre SGF possibilita la lectura d'una cosa que no existeix (ja que s'ha arribat al final) i com a resultat ens comunica la situació produïda. En aquest cas, la variable `r` conté un valor indefinit que no hem de tractar.

Funcionament de la lectura en algun SGF

En algun SGF és obligatori estar segur de no haver arribat al final del fitxer per a poder efectuar la lectura. En cas d'estar al final del fitxer i efectuar una lectura, el SGF dóna un error greu que pot arribar a provocar l'avortament de l'execució del programa.

En aquests SGF hi ha una funció similar a:

```
funció final_fs (f:fitxer_seq de T) retorna lògic
```

La funció retorna `cert` si s'ha arribat al final del fitxer i `fals` en cas contrari, la qual cosa permet saber si es pot fer o no la lectura.

Ara bé, hi ha SGF que funcionen tal com hem considerat més amunt i també disposen de la funció `final_fs`. És a dir, el fet que un SGF aporti una funció `final_fs` no implica que s'hagi de comprovar el final abans de fer una lectura.

4) Funció `escriure_fs`

```
funció escriure_fs (f:fitxer_seq de T, r:T) retorna enter;
```

La funció `escriure_fs` efectua l'escriptura lògica del registre `r` (segon argument) cap el buffer. El funcionament no diferit de l'enregistrament fa que la funció `escriure_fs` provoqui sempre una escriptura física i, per tant, que puguem saber si l'enregistrament ha estat o no correcte. En cas de tenir activat l'enregistrament diferit, les dades queden en el buffer fins que al SGF ho creu oportú (pot ser després de moltes altres instruccions i fins i tot en el moment de tancar el fitxer).

5) Funció `modificar_fs`

```
funció modificar_fs (f:fitxer_seq de T, r:T) retorna enter;
```

La funció `modificar_fs` efectua la modificació d'un registre existent al fitxer que ha d'haver estat prèviament llegit o modificat. És a dir, per a modificar un registre cal estar-hi situat al damunt, cosa que s'aconsegueix havent-lo llegit prèviament. Quan es modifica s'hi continua estant situat al damunt, per la qual cosa es pot tornar a modificar. Quan s'executa una altra lectura es passa a estar situat damunt el registre següent.

En aquests moments coneixem les operacions genèriques per al tractament de fitxers seqüencials i les operacions que en particular presenta el llenguatge C. Ara presentarem algorismes típics de tractament de fitxers seqüencials, primer en pseudocodi i posteriorment aplicats en llenguatge C.

Els algorismes en pseudocodi parteixen de les premisses següents:

- Gestionarem un fitxer seqüencial de nom `FITXER.DAT` que emmagatzema registres d'un tipus `T` definit.

- El tipus `T` conté, almenys, un camp de nom `k` i de tipus `TK`, el qual considerarem identificador dels registres del tipus `T` en els casos que ens convingui.

- Considerarem l'existència de les accions següents prèviament dissenyades pel programador en funció dels tipus `T` i `TK`:

```
acció llegir_T (var r: T);
/* Demana, per teclat, les dades d'una instància r de tipus T, validant els valors
introduïts */

acció llegir_k (var k: TK);
/* Demana, per teclat, un valor corresponent al camp k de tipus TK, validant el valor
introduït */

acció llegir_T_menys_k (k: TK, var r: T);
/* Demana, per teclat, les dades d'una instància r de tipus T excepte la del camp k,
validant els valors introduïts i deixant la variable r plena amb les dades introduïdes
i amb el valor k passat per paràmetre */

acció visualitzar_p_T (r: T);
/* Visualitza una instància r de tipus T ocupant la pantalla */

acció visualitzar_f_T (fila: natural, r: T);
/* Visualitza una instància r de tipus T en una fila de la pantalla */

acció copiar_persona (var destí: T, origen: T);
/* Copia el contingut de la instància origen a la instància destí */
```

- Considerarem, també, aquestes altres accions:

```
acció missatge (s: cadena);
/* Presenta el contingut de cadena i espera que l'usuari premi una tecla, moment en
què desapareix el missatge */

acció pregunta (s: cadena, var resp: caràcter, valors: cadena);
/* Presenta el contingut de cadena en pantalla i espera un caràcter introduït per
teclat, el qual transforma a majúscula si és possible i obliga que coincideixi amb
algun dels caràcters de la cadena valors */

acció avortar_programa;
/* Aquesta acció provoca l'avortament del programa. Té la mateixa utilització que la
funció exit del llenguatge C */

acció avortar_acció;
/* Aquesta acció provoca l'avortament de l'acció en execució i continua l'execució en
el lloc on s'havia produït la crida; en les funcions no és necessària, ja que el seu
equivalent és utilitzar una sentència retorna en el moment en què es vol abandonar la
funció */
```

Recordem que en el tractament de fitxers és necessari anar controlant els possibles errors o, millor dit, els valors que retornen les funcions, els quals haurem d'interpretar per a decidir si cal considerar-los o no error i actuar en conseqüència.

Recordem que els valors enters retornats indiquen les diferents anomalies produïdes en l'execució de l'operació. Utilitzarem constants simbòliques com `FITXER_INEXISTENT`, `FI_FITXER`... per a referir-nos als diferents valors retornats.

Considerem l'acció següent:

```
acció error (num: natural, arx: cadena, op: cadena) és
   escriure ("Error ", num, " sobre el fitxer ", arx, " en l'operació ", op);
   esperar_tecla;
fiacció
```

Utilitzarem aquesta acció per a visualitzar els errors detectats i comprovar que no es corresponguin amb cap de les situacions esperades.

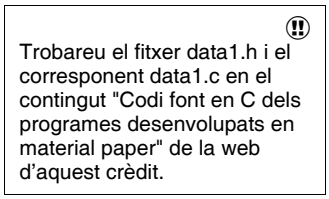
Per dur a la pràctica en llenguatge C els algorismes que anirem presentant en pseudocodi, es treballarà amb els tipus de dades i funcions definits en els fitxers capçalera `generic1.h`, `data1.h` i `persona1.h` que reproduïm a continuació.

```
/* generic1.h */

#ifndef __GENERIC__H__
#define __GENERIC__H__

void missatge_ae (char *s);
/*  Funció que treu un missatge a la darrera línia de la pantalla
    i espera a que l'usuari premi una tecla per continuar
      Paràmetres:
         s   Missatge a visualitzar
*/

void missatge_se (char *s);
/* Funció que treu un missatge a la darrera línia de la pantalla
      Paràmetres:
         s   Missatge a visualitzar
*/

void netejar_missatge(void);
/* Funció que esborra la darrera línia de la pantalla, destinada a missatges */

void pregunta (char *s, char *resp, char *valors);
/* Funció que fa una pregunta a la darrera línia de la pantalla
      i espera una resposta d'un sol caràcter, el qual és validat per
      la mateixa funció
      Paràmetres:
```

Trobareu el fitxer generic1.h i el corresponent generic1.c en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
        s       Text de la pregunta
        resp    Variable de tipus caràcter que queda plena amb la resposta
        valors  Conjunt de caràcters com a possible solució
*/

#endif
```

```
/* data1.h */

#ifndef __DATA__H__
#define __DATA__H__

struct t_data
{
   unsigned char dia, mes;
   unsigned short any;
};

int esAnyTraspas (int any);
/* Comprova si el valor de l'argument correspon o no a un any de traspàs
   Retorna: 0: No és any de traspàs
            1: Sí és any de traspàs
*/

int dataOK (int dia, int mes, int any);
/* Comprova si la combinació dels arguments "dia", "mes" i "any" corresponen
   a una data correcta.
   Retorna: 0: No és data correcta
            1: Sí és data correcta
*/

int inputData (struct t_data *pdata);
/* Efectua, per l'entrada estàndard, la lectura de la data en format dd-mm-aaaa
   on el separador pot ser '.', '-' o '/'.
   Significat dels arguments:
      pdata: Variable que recollirà la data introduïda per l'usuari
   Significat del valor retornat:
      0: Data incorrecta - La variable manté el valor inicial
      1: Data correcta
*/

void outputData (struct t_data data);
/* Visualitza, per la sortida estàndard, la data en format dd-mm-aaaa */

int getDia (struct t_data data);
/* Retorna el dia de la data de l'argument */

int getMes (struct t_data data);
/* Retorna el mes de la data de l'argument */

int getAny (struct t_data data);
/* Retorna l'any de la data de l'argument */

void dataToCadena (char *cadena, struct t_data data);
/* Efectua la conversió d'una data a cadena, en format dd-mm-aaaa
   Significat dels arguments:
      data  : Variable que conté la data a convertir
      cadena: Variable que recull la conversió de la data
```



```
                És responsabilitat del programador que tingui capacitat per a
                10 caràcters (més 1 per '\0')
*/

int cadenaToData (char *cadena, struct t_data *pdata);
/* Omple una variable de tipus data a partir del contingut d'una cadena, que
    es suposa conté la data en format dd-mm-aaaa on el separador pot ser '.',
    '-' o '/'.
   Significat dels arguments:
     pdata -> Variable que recollirà la data existent a la cadena
   Significat del valor retornat:
     0 -> Data incorrecta - La variable pdata manté el valor inicial
     1 -> Data correcta
*/

int compararDates (struct t_data data1, struct t_data data2);
/* Efectua la comparació de dues dates, amb els resultats:
     0: Són iguals
     -1: data1 és menor que data 2
     +1: data1 és major que data 2
*/

int diesEntreDates (struct t_data data1, struct t_data data2);
/*  Efectua la diferència de data1 menys data2, obtenint com a resultat el
    número de dies que hi ha entre les dues:
     S'obté el resultat 0 si data1==data2
     S'obté un número positiu si data1 > data2
     S'obté un número negatiu si data1 < data2
*/

struct t_data dataSumaDies (struct t_data data, int numDies);
/* Retorna una data corresponent al resultat de sumar els dies de l'argument
   "numDies" a la data de l'argument "data".
   L'argument "numDies" pot ser positiu, negatiu o zero, i per tant, el resultat
   serà una data posterior, igual o anterior a la data de l'argument "data"
*/
#endif
```

```
/* persona1.h */

#ifndef __PERSONA__H__
#define __PERSONA__H__

#include "data1.h"

struct t_persona
{
   char nif[10];
   char cognom1[21], cognom2[21], nom[16];
   struct t_data data_naix;
   double pes;
   unsigned char alt;   /* Sense decimals - Ningú passa dels 255 cm, no? */
};

void pantallaPersona();
/* Situa els títols dels diferents camps en una pantalla */

void inputPersona (struct t_persona *p);
```

‼ Trobareu el fitxer persona1.h i el corresponent persona1.c en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* Efectua la lectura de les dades d'una persona per l'entrada estàndard */

void outputPersonaFor (struct t_persona p);
/* Visualitza una persona per la sortida estàndard, en format formulari */

void outputPersonaTab (unsigned char fil, struct t_persona p);
/* Visualitza una persona per la sortida estàndard, en format tabular,
   a la fila que indica l'argument "fil" */

char *getPersonaNif (struct t_persona *p);
/* Retorna el Nif de la persona */

char *getPersonaCognom1 (struct t_persona *p);
/* Retorna el Cognom1 de la persona */

char *getPersonaCognom2 (struct t_persona *p);
/* Retorna el Cognom2 de la persona */

char *getPersonaNom (struct t_persona *p);
/* Retorna el Nom de la persona */

struct t_data getPersonaDataNaix (struct t_persona *p);
/* Retorna la Data de Naixement de la persona */

double getPersonaPes (struct t_persona *p);
/* Retorna el pes de la persona */

int getPersonaAlt (struct t_persona *p);
/* Retorna l'alçada de la persona */

int setPersonaNif (struct t_persona *p, char *nouNif);
/* Intenta modificar el NIF de la persona.
   Retorna 1: Nou NIF correcte
        0: Nou NIF erroni - No s'ha modificat
*/

int setPersonaCognom1 (struct t_persona *p, char *nouCognom1);
/* Intenta modificar el Cognom1 de la persona.
   Retorna 1: Nou Cognom1 correcte
        0: Nou Cognom1 erroni - No s'ha modificat
*/

int setPersonaCognom2 (struct t_persona *p, char *nouCognom2);
/* Intenta modificar el Cognom2 de la persona.
   Retorna 1: Nou Cognom2 correcte
          0: Nou Cognom2 erroni - No s'ha modificat
*/

int setPersonaNom (struct t_persona *p, char *nouNom);
/* Intenta modificar el nom de la persona.
   Retorna 1: Nou nom correcte
           0: Nou nom erroni - No s'ha modificat
*/

int setPersonaDataNaix (struct t_persona *p, struct t_data novaDataNaix);
/* Intenta modificar la Data de Naixement de la persona.
   Retorna 1: Nova data correcta
        0: Nova data errònia - No s'ha modificat
```

```
*/

int setPersonaPes (struct t_persona *p, double nouPes);
/* Intenta modificar el pes de la persona.
   Retorna 1: Nou pes correcte
           0: Nou pes erroni - No s'ha modificat
*/

int setPersonaAlt (struct t_persona *p, int novaAlt);
/* Intenta modificar l'alçada de la persona.
   Retorna 1: Nova alçada correcta
           0: Nova alçada errònia - No s'ha modificat
*/
#endif
```

3.2. Introducció de registres

L'algorisme més simple corresponent a la introducció de registres en un fitxer seqüencial consisteix a anar afegint registres pel final del fitxer sense cap tipus de comprovació sobre el contingut de les dades que s'afegeixen.

Per a poder afegir registres caldrà obrir el fitxer en mode extensió. Què cal fer si el fitxer no existeix? Tenim dues opcions: crear-lo i afegir registres o avisar l'usuari de la situació produïda i acabar l'execució del programa.

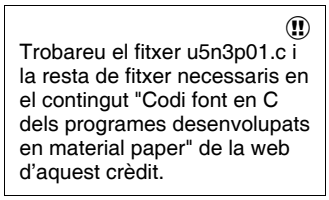
L'algorisme en pseudocodi és aquest:

```
programa introduir_dades és
   var  f: fitxer_seq de T;  /* fitxer intern */
        x: enter; /* per a controlar l'execució de les operacions amb el fitxer */
        r: T;/* variable registre per a omplir i gravar al fitxer */
        opc: caràcter;
   fivar
   x = obrir_fs (f, "FITXER.DAT", 'E');
   /* obrir en extensió per a afegir pel final */
   opció
      cas x == FITXER_INEXISTENT:
         pregunta ("El fitxer FITXER.DAT no existeix. Vol crear-lo?", opc, "SN");
         si opc == 'S' llavors
            x = obrir_fs (f, "FITXER.DAT", 'I'); /* creació del fitxer */
            si x != 0 llavors
               error (x, "FITXER.DAT", "creació");
               avortar_programa;
            fisi
         sinó avortar_programa;
         fisi
      cas x != 0:
         error (x, "FITXER.DAT", "obrir en extensió");
         avortar_programa;
   fiopció
   /* El fitxer FITXER.DAT és obert en situació d'afegir registres */
```

```
   fer
      llegir_T ( r );   /* L'usuari omple les dades del registre */
      pregunta ("Vol gravar-lo?", opc, "SN");
      si opc == 'S' llavors
         x = escriure_fs (f, r);
         si x != 0 llavors
            error (x, "FITXER.DAT", "escriure registre");
         fisi
      fisi
      pregunta ("Vol afegir més registres?", opc, "SN");
   mentre opc == 'S';
   x = tancar_fs (f);
   si x != 0 llavors
      error (x, "FITXER.DAT", "tancar");
   fisi
fiprograma
```

Vegem-ne la versió en llenguatge C utilitzant la funció `fwrite` per a enregistrar un registre sencer. Abans, però, cal tenir en compte que el mode d'obertura `a` en el llenguatge C crea el fitxer si aquest no existeix. L'algorisme anterior controla el fet que el fitxer no existeixi i en aquest cas demana a l'usuari l'oportuna confirmació per a crear-lo.

Vegem la versió en llenguatge C sobre un fitxer de nom `PERSONES.DAT` on enregistrem dades del tipus `persona` definit al fitxer `persona1.h`.



Trobareu el fitxer u5n3p01.c i la resta de fitxer necessaris en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* u5n3p01.c */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "verscomp.h"
#include "generic1.h"
#include "persona1.h"

void main()
{
   FILE *f;
   struct t_persona p;
   char opc;

   clrscr();
   if (existeix_fitxer ("PERSONES.DAT")==-1)
   {
      pregunta ("El fitxer PERSONES.DAT no existeix. Vol crear-lo?", &opc, "SN");
      if (opc=='N') exit (0);
   }
   if ((f = fopen ("PERSONES.DAT","ab")) == NULL)
   {
      missatge_ae ("Error en obrir el fitxer PERSONES.DAT");
      exit (1);
   }
   /* El fitxer PERSONES.DAT està obert en situació d'afegir registres */
   do
   {  inputPersona (&p);
```

```
      pregunta ("Vol gravar-la?", &opc, "SN");
      if (opc == 'S')
      {
         fwrite (&p, sizeof(struct t_persona), 1, f);
         if (ferror(f))
         {
            missatge_ae ("Error en gravar registre al fitxer PERSONES.DAT");
            clearerr(f);
         }
      }
      pregunta ("Vol afegir més registres?", &opc, "SN");
   } while (opc == 'S');
   if (fclose(f)) missatge_ae ("Error en tancar el fitxer PERSONES.DAT");
   clrscr();
}
```

Executeu aquest programa i observeu posteriorment, amb l'ordre de sistema operatiu que correspongui (`type` per a DOS/Windows, `cat` per a Unix/Linux), el contingut del fitxer. Podreu llegir les dades corresponents a les cadenes, tot i que heu de tenir present que l'espai de la cadena posterior al '\0' també queda enregistrat i, per tant, possiblement veureu "porqueria". Les dades numèriques, però, queden enregistrades en format binari i, per tant, no es poden reconèixer a simple vista. Per acabar, observeu també la grandària del fitxer: en Turbo C coincideix amb n * 81, ja que 81 és la llargada de l'estructura `t_persona` i n és el nombre de registres arxivats. En canvi en gcc coincideix amb n * 84... Com és possible si la suma dels bytes de cadascun dels camps de l'estructura `t_persona` és de 81 bytes (10 del nif + 21*2 del cognoms + 16 del nom + 4 de la data (1 del dia + 1 del mes + 2 de l'any) + 8 del pes + 1 de l'alçada)?

Per què `sizeof (struct xxx)` en gcc retorna valor major que la suma de les llargades dels camps que formen l'estructura?

No es tracta d'un `bug` (error) del compilador! El compilador gcc, que genera codi de 32 bits, per qüestions de millorar el rendiment en temps d'execució, pot afegir alguns bits a certes estructures de manera que la seva longitud en memòria sigui òptima per al rendiments dels accessos, de manera que la llargada de l'estructura és superior que la llargada dels camps que la componen.

En ocasions es pot intentar minimitzar aquest farciment de bits redissenyant l'estructura i situant els camps més llargs abans que els més curs, però això no sempre assegura que la longitud de l'estructura coincideixi amb la suma de les longituds dels camps. A més, no sempre és factible canviar el disseny de l'estructura.

Es pot forçar, però, que gcc actuï sense farcir l'estructura amb bits. Hi ha dues maneres:

a) Afegint al final de l'estructura la definició `__atribute__((packed))` com es veu a continuació:

```
struct t_persona
{
   char nif[10];
   char cognom1[21], cognom2[21], nom[16];
   struct t_data data_naix;
   double pes;
```

```
    unsigned char alt;
} __attribute__((packed));
```

Aquesta opció, però, no és possible en C++. Únicament és vàlida en fonts C.

b) Des de la versió 2.7.0, gcc aporta l'opció `fpack-struct` que tracta totes les estructures utilitzades en el programa com si portessin la definició `__atribute__((packed))`. L'opció funciona correctament a partir de la versió 2.96 de gcc.
Aquesta opció, però, només es pot utilitzar si no s'ha d'utilitzar cap de les estructures definides en les llibreries de funcions que aporta gcc, doncs aquestes s'ha compilat sense l'opció `fpack-struct`. Per altra banda, comentar que l'opció

Aquesta situació pot portar problemes quan les dades gestionades per estructures s'enregistren a disc des de l'estructura (via `fwrite`) o es llegeixen de disc a l'estructura (via `fread`), doncs a disc hi queden els bits de farciment que ha afegit gcc. De fet això no és un problema si el fitxer només es gestiona amb aplicacions desenvolupades per gcc, però, pot produir problemes si el fitxer és gestionat per diferents aplicacions. Així, en el nostre exemple, tindríem problemes si el mateix fitxer PERSONES.DAT volem que sigui gestionat pel programa generat amb Turbo C i pel programa generat amb gcc. El problema no existeix si en el cas de gcc utilitzem l'afegit `__atribute__((packed))`.

De fet, la manera més robusta i portable de llegir i esciure estructures és via un buffer de caràcters, de manera que els valors de l'estructura a gravar a disc es passin, camp a camp, al buffer de caràcters per posteriorment enregistrar a disc aquest buffer i que els valors a llegir de disc cap a una estructura es llegeixin del disc cap al buffer i d'aquí, camp a camp, cap a l'estructura.

És important notar el fet que hem obert el fitxer en mode binari, cosa que ens assegura que cada registre ocupi exactament 81 bytes (o 84 en gcc). Si l'obrim en mode text podem tenir sorpreses, ja que els valors `'\n'` són transformats en dos caràcters i podem tenir valors `'\n'` en els residus de les cadenes ("porqueria" posterior al `'\0'`) o en els valors numèrics en format binari.

La utilització de la funció `fwrite` i del mode binari ens assegura que tots els registres queden emmagatzemats amb la mateixa longitud i, per tant, la recuperació de la informació també serà senzilla. Ara bé, segur que enregistrem "porqueria" (els trossos de cadenes posteriors al `'\0'`).

Aprofitem al màxim l'espai

Per tal d'aprofitar al màxim l'espai del disc podem enregistrar només els caràcters amb significat, és a dir, estalviar-nos l'enregistrament de la "porqueria". En aquest cas, el programa es complica, ja que haurem de gravar el registre camp a camp marcant d'alguna manera el final de les cadenes perquè no sempre tindran la mateixa longitud.

Sembla lògic, per tant, dissenyar una funció que gravi el registre a trossos. Però, i si es produeix un error en l'enregistrament d'algun tros? En aquesta situació tindríem el registre enregistrat a mitges. Caldria esborrar el que havíem gravat? Com es fa, això? Controlar-ho no és gens senzill i potser no val la pena complicar-nos la vida si disposem de la funció fwrite, que ens aporta la funcionalitat necessària.

A més, de cara a la pròxima gestió de fitxers relatius, ens interessa tenir tots els registres de la mateixa longitud.

3.3. Recorregut

L'algorisme per a recórrer un fitxer seqüencial és similar a l'algorisme per a recórrer una taula. Es comença pel principi i es va avançant posició a posició fins a detectar el final.
Així doncs, l'algorisme en pseudocodi és aquest:

```
programa recorregut és
   var f: fitxer_seq de T;/* fitxer intern */
       x: enter; /* per a controlar l'execució de les operacions amb el fitxer */
       r: T; /* variable registre */
   fivar
   x = obrir_fs (fv, "FITXER.DAT", 'C');  /* obertura del fitxer en consulta */
   si x != 0 llavors
      error (x, "FITXER.DAT", "obrir en consulta");
      avortar_programa;
   fisi
   /* Tenim el fitxer obert */
   x = llegir_fs (f, r);
   mentre x == 0 fer
      tractament(r);    /* procés a efectuar amb el registre */
      x = llegir_fs (f, r);
   fimentre
   si x != FI_FITXER llavors
      error (x, "FITXER.DAT", "llegir registre");
   fisi
   tancar_fs (f);
fiprograma
```

Vegem-ne una aplicació en llenguatge C. Visualitzem ara per pantalla les persones existents al fitxer `PERSONES.DAT`.

```
/* u5n3p02.c */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "generic1.h"
#include "persona1.h"

void main()
{
   FILE *f;
   struct t_persona p;

   clrscr();
   if ((f = fopen ("PERSONES.DAT","rb")) == NULL)
   {
     missatge_ae ("No es pot obrir el fitxer PERSONES.DAT en lectura. ");
      exit (1);
   }
   /* Tenim el fitxer obert */
   fread (&p, sizeof(struct t_persona), 1, f);
   while (!ferror(f) && !feof(f))
   {
```


Trobareu el fitxer u5n3p02.c i la resta de fitxers necessaris en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
      outputPersonaFor (p);
      missatge_ae ("");
      fread (&p, sizeof(struct t_persona), 1, f);
   }
   if (ferror(f))
      missatge_ae ("Error en llegir registre del fitxer PERSONES.DAT. ");
   fclose (f);
   clrscr();
}
```

3.4. Modificació amb còpia

Aquest és el típic algorisme que cal utilitzar quan s'ha d'efectuar una modificació a tots els registres d'un fitxer seqüencial per tal d'aconseguir un nou fitxer seqüencial (la còpia) amb els registres modificats. Assignarem el nom `FITXER.DAT` al fitxer que cal modificar i el nom `AUXILIAR.DAT` al nom del nou fitxer.

Vegem els casos en què és necessari aquest algorisme:

- Per a canviar l'estructura dels registres del fitxer: afegir un nou camp, modificar el domini de valors d'un camp –més o menys grandària–, eliminar un dels camps existents. En aquest cas els dos fitxers serien de diferents tipus (T1 i T2).

- Per a regenerar el fitxer quan s'està gestionant l'esborrat de registres amb una marca, la qual cosa implica l'existència de "forats" que amb aquest procés poden ser eliminats, ja que es llegiran de `FITXER.DAT` i no s'enregistraran en `AUXILIAR.DAT`.

També hi ha un cas en què és utilitzable –però no necessari– aquest algorisme:

- Per a efectuar una transformació de valor en algun(s) camp(s) per a tots els registres del fitxer mantenint el fitxer original. En particular, observeu que si no hi ha cap transformació a realitzar, l'algorisme consisteix a efectuar una còpia del fitxer original.

Ara dissenyarem l'algorisme en pseudocodi per al cas en què els dos fitxers tenen la mateixa estructura i en què tots els registres del fitxer `FITXER.DAT` són traspassables al fitxer `AUXILIAR.DAT`. Amb poques modificacions es pot aconseguir el disseny de l'algorisme mitjançant l'eliminació de registres esborrats amb marca o mitjançant la modificació de l'estructura de registre.

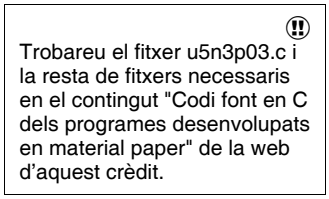
```
programa modificació_amb_còpia és
   var  fv, fn: fitxer_seq de T;  /* fitxers interns */
```

```
        x: enter; /* per a controlar l'execució de les operacions amb els fitxers */
        rv, rn: T; /* variables registre */
   fivar
   x = obrir_fs (fv, "FITXER.DAT", 'C');  /* obertura del fitxer vell en consulta */
   si x != 0 llavors
      error (x, "FITXER.DAT", "obrir en consulta");
      avortar_programa;
   fisi
   x = obrir_fs (fn, "AUXILIAR.DAT", 'I');   /* creació del nou fitxer */
   si x != 0 llavors
      error (x, "AUXILIAR.DAT", "creació");
      tancar_fs (fv);
      /* tanquem els fitxers oberts abans d'avortar el programa */
      avortar_programa;
   fisi
   /* Tenim els dos fitxers oberts */
   x = llegir_fs (fv, rv);
   mentre x == 0 fer
      rn = transformació (rv);  /* procés a efectuar sobre el registre vell */
      x = escriure_fs (fn, rn);
      si x != 0 llavors
         error (x, "AUXILIAR.DAT", "escriure registre");
         tancar_fs (fv); tancar_fs (fn);
         missatge ("Eliminem el fitxer generat fins al moment.");
         sistema (esborrar "AUXILIAR.DAT"); avortar_programa;
      fisi
      x = llegir_fs (fv, rv);
   fimentre
   si x != FI_FITXER llavors
      error (x, "FITXER.DAT", "llegir registre");
      tancar_fs (fv); tancar_fs (fn);
      missatge ("Eliminem el fitxer generat fins al moment.");
      sistema (esborrar "AUXILIAR.DAT");
   sinó
      tancar_fs (fv);    /* no controlem l'error; no s'ha modificat el fitxer */
      x = tancar_fs (fn); /* el fitxer nou sí que s'ha modificat */
      si x != 0 llavors
         error (x, "AUXILIAR.DAT", "tancar");
         missatge ("Verifiqui l'estat del fitxer AUXILIAR.DAT");
         /* També hauríem pogut eliminar el fitxer */
      fisi
   fisi
fiprograma
```

Observem que els errors greus provoquen l'avortament del programa i l'eliminació del fitxer generat fins al moment, ja que és inconsistent. Observem, també, que l'eliminació del fitxer es fa amb la utilització de la corresponent crida de sistema, cosa que provoca la manca de transportabilitat de l'algorisme entre diferents sistemes. Per aquest motiu, caldrà emprar, si existeix, la instrucció d'esborrament de fitxers que proporcioni el llenguatge.

Vegem-ne una aplicació en llenguatge C. Donat el fitxer `PERSONES.DAT` (registres de longitud fixa enregistrats amb la funció `fwrite`), es vol aconseguir un fitxer còpia `AUXILIAR.DAT` en el que els DNI de tots els

registres tinguin una llargada de nou caràcters, omplint amb zeros, per l'esquerra, aquells que tinguin menys longitud.




```
/* u5n3p03.c */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include "generic1.h"
#include "persona1.h"

void main()
{
   FILE *fv, *fn;
   struct t_persona p;

   clrscr();
   if ((fv = fopen ("PERSONES.DAT","rb")) == NULL)
   {
     missatge_ae ("No es pot obrir el fitxer PERSONES.DAT en lectura. ");
      exit (1);
   }
   if ((fn = fopen ("AUXILIAR.DAT","wb")) == NULL)
   {
      missatge_ae ("Error en crear el fitxer AUXILIAR.DAT. ");
      fclose (fv); exit (1);
   }
   /* Els dos fitxers estan oberts */
   fread (&p, sizeof(struct t_persona), 1, fv);
   while (!ferror(fv) && !feof(fv))
   {
      if ( strlen(p.nif) < 9 )
      {
         int longitud=strlen(p.nif);
         int i;
         for (i=9; longitud>=0; i--,longitud--) p.nif[i]=p.nif[longitud];
         for (; i>=0; i--) p.nif[i]='0';
      }
      fwrite (&p, sizeof(struct t_persona), 1, fn);
      if (ferror(fn))
      {
         missatge_ae ("Error en gravar registre al fitxer AUXILIAR.DAT. ");
         missatge_ae ("Eliminem el fitxer AUXILIAR.DAT generat fins el moment. ");
         fclose (fv); fclose (fn);
         unlink ("AUXILIAR.DAT"); exit (1);
      }
      fread (&p, sizeof(struct t_persona), 1, fv);
   }
   if (ferror(fv))
   {
      missatge_ae ("Error en llegir registre del fitxer PERSONES.DAT. ");
      missatge_ae ("Eliminem el fitxer AUXILIAR.DAT generat fins el moment. ");
      fclose (fv); fclose (fn);
      unlink ("AUXILIAR.DAT");
   }
   else
   {
```

```
      fclose (fv);
      if (fclose(fn))
      {
         missatge_ae ("Error en tancar el fitxer AUXILIAR.DAT. ");
         missatge_ae ("Verifiqui l'estat del fitxer AUXILIAR.DAT. ");
      }
   }
   clrscr();
}
```

En aquest programa en llenguatge C hem utilitzat una funció del llenguatge que no havia aparegut fins ara: `unlink`. En els diferents programes que treballem en aquest llibre comentarem les funcions que vagin apareixent. Ara bé, en aquest moment hauríeu d'haver adquirit la capacitat de cercar, en el manual de referència del llenguatge C –sistema d'ajuda en línia–, les funcions que el llenguatge aporta i que ens puguin ser útils en la codificació dels algorismes.

```
#include <stdio.h>
int unlink (const char *nom_fitxer)
```

Aquesta funció elimina el fitxer `nom_fitxer`. Cal tenir accés al fitxer perquè es pugui executar amb èxit. En cas de no tenir permisos d'accés, la funció `chmod` (us sona?) els permet assolir.

El llenguatge C aporta també la macro `remove`, que també permet l'eliminació d'un fitxer. Aquesta macro consisteix en una crida de la funció `unlink`. Per tant, és millor utilitzar directament aquesta funció.

3.5. Modificació sense còpia

Aquest és el típic algorisme que cal utilitzar quan s'ha d'efectuar una modificació a tots els registres d'un fitxer seqüencial sobre el mateix fitxer. L'algorisme és semblant al que hem vist en l'apartat anterior, però amb una diferència important: es treballa amb un únic fitxer.

```
programa modificació_sense_còpia és
   var f: fitxer_seq de T;/* fitxer intern únic */
       x: enter; /* per a controlar l'execució de les operacions amb el fitxer */
       r: T; /* variable registre */
   fivar
   x = obrir_fs (fv, "FITXER.DAT", 'A');  /* obertura del fitxer en actualització */
   if x != 0 llavors
      error (x, "FITXER.DAT", "obrir en actualització");
      avortar_programa;
   fisi
   /* Tenim el fitxer obert */
   x = llegir_fs (f, r);
   mentre x == 0 fer
      r = transformació (r);    /* procés a efectuar sobre el registre */
```

```
      x = modificar_fs (f, r);
      si x != 0 llavors
         error (x, "FITXER.DAT", "modificar registre");
         tancar_fs (f);
         missatge ("Verifiqui l'estat del fitxer.");
         avortar_programa;
      fisi
      x = llegir_fs (f, r);
   fimentre
   si x != FI_FITXER llavors
      error (x, "FITXER.DAT", "llegir registre");
   fisi
   x = tancar_fs (f); /* hi pot haver gravacions pendents */
   si x != 0 llavors
      error (x, "FITXER.DAT", "tancar");
      missatge ("Verifiqui l'estat del fitxer. ");
   fisi
fiprograma
```

A diferència de l'algorisme "Modificació amb còpia", en què els errors greus provocaven l'avortament del programa amb l'eliminació prèvia del fitxer generat fins al moment, en aquest cas el fitxer no es pot eliminar perquè es tracta del fitxer original. Això vol dir que, en funció de l'error produït, el fitxer pot quedar en mal estat. Ja feu còpies de seguretat?

Còpies de seguretat

És molt important tenir còpia de seguretat dels fitxers que contenen les dades, ja que en qualsevol moment pot produir-se un error causat per:

- un deteriorament del suport d'emmagatzematge,
- una fallada del programa,
- una fallada de l'usuari.

Vegem-ne una aplicació en llenguatge C. Volem repetir el procés de modificació de DNI omplint-lo amb zeros per l'esquerra fins a tenir nou caràcters de llargada, però sobre el mateix fitxer `PERSONES.DAT`.

Abans, però, hem de comentar com s'aconsegueix una modificació en un fitxer seqüencial en C. Fixeu-vos que hem vist funcions per a escriure. Però on escriuen aquestes funcions? Ho fan a partir del lloc on està situat el punter d'escriptura. I on està situat aquest punter?

D'entrada, cal tenir obert el fitxer en mode `w` o `w+` o `r+`, de manera que s'hi pugui escriure (modificant o afegint). Recordeu que en els modes `a` o `a+` també es pot escriure, però únicament afegint pel final, i que el mode `r` és únicament de lectura. El llenguatge C manté, per cada fitxer, un punter, que marca la posició posterior a la darrera posició a la qual s'ha accedit. Caldrà moure aquest punter per a situar-nos sobre el registre a modificar i escriure-hi al damunt.

Accés directe a bytes del fitxer

Fixeu-vos que ens volem situar en un determinat byte del fitxer. Fer això implica estar utilitzant un accés directe per posició. És lícit això, quan estem treballant amb un accés seqüencial?

Ja hem dit abans que al SGF del llenguatge C és molt rudimentari i no disposa de tractament de registres. Fixeu-vos que estem fent el tractament de registres d'acord amb la utilització de les funcions `fwrite` i `fread`. L'única manera que tenim de modificar un registre és tornar-nos a situar a l'inici del registre i escriure-hi al damunt. Es tracta,

doncs, d'un accés directe a bytes i cal no confondre'l amb el tractament de fitxers relatius (accés seqüencial i directe per posició).

Per a poder situar el punter dins el fitxer, el llenguatge C ens facilita la funció següent:

```
#include <stdio.h>
int fseek (FILE *f, long desp, int pos)
```

Aquesta funció desplaça `desp` bytes (pot ser negatiu) el punter a partir de la posició `pos`, la qual ha de ser una de les constants simbòliques següents:

`SEEK_SET`	Inici del fitxer
`SEEK_CUR`	Posició actual
`SEEK_END`	Final del fitxer

Cal tenir en compte que `desp` ha de ser un valor de tipus `long`. Utilitzar valors de tipus `int` pot provocar errors a l'hora de situar-se a la posició zero (inici) del fitxer.

Aquesta funció retorna un valor zero si ha pogut moure el punter. En cas contrari, retorna un valor diferent de zero.

Una darrera consideració. Quan el fitxer s'obre per a la lectura i l'escriptura, es pot començar a utilitzar qualsevol tipus d'accés (lectures o escriptures), però una vegada efectuat un accés, cal continuar amb el mateix. Si es vol canviar, cal situar obligatòriament el punter amb la funció `fseek`, encara que ens mantinguem en la mateixa posició. Hi ha, però, una excepció: quan efectuant lectures s'arriba al final del fitxer. En aquest cas, immediatament es pot efectuar escriptura sense necessitat d'utilitzar la funció `fseek`.

```
/* u5n3p04.c */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include "generic1.h"
#include "persona1.h"

void main()
{
   FILE *f;
   struct t_persona p;
   size_t lr;

   clrscr();
   lr = sizeof (struct t_persona);
```

Trobareu el fitxer u5n3p04.c i la resta de fitxers necessaris en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
   if ((f = fopen ("PERSONES.DAT","r+b")) == NULL)
   {
      missatge_ae ("No es pot obrir el fitxer PERSONES.DAT en mode r+. ");
      exit (1);
   }
   /* Tenim el fitxer obert */
   fread (&p, lr, 1, f);
   while (!ferror(f) && !feof(f))
   {
      if ( strlen(p.nif) < 9 )
      {
         int longitud=strlen(p.nif);
       int i;
       for (i=9; longitud>=0; i--,longitud--) p.nif[i]=p.nif[longitud];
         for (; i>=0; i--) p.nif[i]='0';
      }
      fseek (f, -(long)lr, SEEK_CUR);
      fwrite (&p, lr, 1, f);
      if (ferror(f))
      {
         missatge_ae ("Error en gravar registre al fitxer PERSONES.DAT. ");
         missatge_ae ("Verifiqui l'estat del fitxer. ");
         fclose (f); exit (1);
      }
      fseek (f, 0L, SEEK_CUR);
      fread (&p, lr, 1, f);
   }
   if (ferror(f))
      missatge_ae ("Error en llegir registre del fitxer PERSONES.DAT. ");
   if (fclose(f))
   {
      missatge_ae ("Error en tancar el fitxer PERSONES.DAT. ");
      missatge_ae ("Verifiqui l'estat del fitxer. ");
   }
   clrscr();
}
```

3.6. Recerca

Una situació de treball típica en fitxers és la recerca. Hi ha diferents situacions en les quals es pot necessitar una recerca. Caldrà aplicar, en cada moment, l'algorisme més eficient.

Les situacions que es poden presentar acostumen a ser el resultat de les diferents combinacions entre els fets següents:

- El fitxer està ordenat o no ho està.
- El fitxer pot tenir o no pot tenir registres amb el(s) valor(s) de recerca repetits.
- S'ha de consultar un o diversos valors.

3.6.1. Recerca d'un valor per un camp

Suposem que volem cercar el valor `v` pel camp `k`. Si no se'ns diu una altra cosa, no sabem si el fitxer està ordenat o no ho està i si té valors repetits o no en té. Haurem de suposar la situació més desfavorable, és a dir, que no està ordenat i que té valors repetits. D'aquesta manera, assolirem un algorisme que també ens servirà per a fitxers ordenats i sense valors repetits.

L'algorisme consisteix en un recorregut total per el fitxer, registre a registre, comprovant si es tracta del registre cercat. En cas de trobar-ne un, caldrà fer el tractament que pertoqui i continuar la recerca.

Dissenyarem l'algorisme en pseudocodi com una acció a la qual es passa per paràmetre el valor v a cercar. En la pràctica, no té per què ser així. Poden donar-se altres situacions, com per exemple que es demani a l'usuari el valor amb un missatge per pantalla i s'esperi la resposta per teclat.

```
acció recerca ( v: TK) és
   var f: fitxer_seq de T;/* fitxer intern */
       x: enter; /* per a controlar l'execució de les operacions amb el fitxer */
       r: T; /* variable registre */
   fivar
   x = obrir_fs (fv, "FITXER.DAT", 'C');  /* obertura del fitxer en consulta */
   if x != 0 llavors
      error (x, "FITXER.DAT", "obrir en consulta");
      avortar_programa;
   fisi
   /* Tenim el fitxer obert */
   x = llegir_fs (f, r);
   mentre x == 0 fer
      si r.k == v llavors
         tractament(r);   /* procés a efectuar amb el registre */
      fisi
      x = llegir_fs (f, r);
   fimentre
   si x != FI_FITXER llavors
      error (x, "FITXER.DAT", "llegir registre");
   fisi
   tancar_fs (f);
fiacció
```

Farem ara una aplicació en llenguatge C. Volem efectuar un programa que cerqui totes les persones (al fitxer `PERSONES.DAT`) que tinguin un determinat nom que haurà de demanar-se a l'usuari.

```
/* u5n3p05.c */

#include <stdio.h>
#include <conio.h>
#include <string.h>

#include "generic1.h"
#include "persona1.h"

void main()
{
   FILE *f;
   struct t_persona p;
   char nom[16],s[80];
   int aux,q=0;

   clrscr();
   if ((f = fopen ("PERSONES.DAT","rb")) == NULL)
   {
     missatge_ae ("No es pot obrir el fitxer PERSONES.DAT en lectura. ");
      exit (1);
   }
   /* Tenim el fitxer obert */
   pantallaPersona();
   do
   { gotoxy (40,14); aux = scanf ("%15s",nom); fflush (stdin); }
   while (aux == 0);
   fread (&p, sizeof(struct t_persona), 1, f);
   while (!ferror(f) && !feof(f))
   {
     if (!strcmp(nom,p.nom)) {q++; outputPersonaFor (p); missatge_ae ("");}
     fread (&p, sizeof(struct t_persona), 1, f);
   }
   if (ferror(f))
     missatge_ae ("Error en llegir registre del fitxer PERSONES.DAT. ");
   else {
        if (q>0)
           sprintf(s,"S'ha trobat %d persones amb nom %s. ",q,nom);
        else
           sprintf(s,"No hi ha cap persona amb nom %s. ",nom);
        missatge_ae (s);
       }
   fclose (f);
   clrscr();
}
```


Trobareu el fitxer u5n3p05.c i la resta de fitxers necessaris en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Observem que estem utilitzant la funció `sprintf`, que encara no havia estat emprada. Aquesta funció té el mateix funcionament que `printf` i `fprintf`, però amb la diferència que la sortida és enviada a una cadena. És a dir, aquesta funció es pot utilitzar per a omplir cadenes a partir de dades emmagatzemades en altres variables mitjançant el format que interessi. De manera similar, també existeix la funció `sscanf`, equivalent a les funcions `scanf` i `fscanf`.

3.6.2. Recerca d'un valor per un camp identificador

Suposem que volem cercar el valor `v` pel camp `k`, el qual sabem que és identificador, és a dir, que no hi pot haver diferents registres amb valors repetits pel camp `k`.

Al contrari que en l'apartat anterior, ara sabem que no hi pot haver diferents registres amb el mateix valor a cercar `v`, però no tornem a saber res sobre l'ordre dels registres segons el camp `k`. L'algorisme té una diferència important respecte al de l'apartat anterior. En aquest cas, el procés repetitiu de recerca no solament finalitza a causa d'una situació d'error o perquè s'hagi arribat a la fi del fitxer, sinó que també cal acabar-lo si es troba un registre amb el valor cercat, ja que, si no hi pot haver altres registres amb el mateix valor, no cal continuar la recerca.

Així doncs, l'algorisme és aquest:

```
acció recerca_ident ( v: TK) és
   var f: fitxer_seq de T;/* fitxer intern */
       x: enter; /* per a controlar l'execució de les operacions amb el fitxer */
       r: T; /* variable registre */
   fivar
   x = obrir_fs (fv, "FITXER.DAT", 'C');  /* obertura del fitxer en consulta */
   if x != 0 llavors
      error (x, "FITXER.DAT", "obrir en consulta");
      avortar_programa;
   fisi
   /* Tenim el fitxer obert */
   x = llegir_fs (f, r);
   mentre x == 0 i r.k <> v fer
      x = llegir_fs (f, r);
   fimentre
   opció
      cas x == 0: tractament(r); /* procés a efectuar amb el registre */
      cas x == FI_FITXER: missatge ("No hi ha cap registre amb el valor cercat");
      altrament: error (x, "FITXER.DAT", "llegir registre");
   fiopció
   tancar_fs (f);
fiacció
```

Per a aplicar aquest algorisme construirem, amb el llenguatge C, un fitxer de nom `PERS_NIF.DAT` que contingui persones, de manera que el camp `NIF` sigui identificador. En els programes fets fins ara, el fitxer `PERSONES.DAT` contenia registres de tota mena, sense tenir en compte el valor del `NIF`. És bastant lògic considerar que el camp `NIF` sigui un atribut identificador.

És evident que per a construir el fitxer `PERS_NIF.DAT`, a cada alta caldrà efectuar una recerca del valor del `NIF` que es vol donar d'alta per

tal de comprovar que no existeixi en el fitxer aquest registre. A més, és de calaix que l'algorisme ha de demanar, en primer lloc, el `NIF` de la persona que es vol donar d'alta, i abans de sol·licitar cap altra dada, efectuar la recerca. Tindríem un disseny dolent si la recerca s'efectués després de demanar totes les dades a l'usuari.

Així doncs, per a efectuar aquest programa utilitzarem les funcions que ens convinguin de les existents al fitxer `persona1.h` i altres funcions noves, com:

- Funció que permeti efectuar la lectura, per teclat, d'un nif.

  Aquesta funcionalitat es pot veure independent del concepte de persona, doncs ens podem trobar amb el concepte nif en altres entitats (empreses, per exemple). Per aquest motiu, pot ser bó dissenyar la següent funció dins el paquet de les funcions genèriques (`generic1.h`), utilitzables en diversos programes:

```
void inputNif (char *nif);
/* Efectua, per teclat, l'entrada del nif, el qual és omplert, si fa falta, amb zeros
per l'esquerra fins a tenir 9 caràcters de longitud */
```

- Funció que permeti efectuar la lectura, per teclat, de totes les dades d'una persona a excepció del nif.

  La ubicació lògica d'aquesta funció és en el paquet que inclou totes les funcions per a gestionar persones (`persona1.h`):

```
void inputPersonaSenseNif (struct t_persona *p);
/* Efectua l'entrada de les dades d'una persona exceptuant el nif, en la pantalla
   dissenyada per a efectuar l'entrada de dades de tota la persona. */
```

- Funció que permeti efectuar la lectura de només el nif en un formulari (pantalla) dissenyat per a continuar amb la lectura de la resta de les dades d'una persona.

  La ubicació lògica d'aquesta funció torna a ser en el paquet que inclou totes les funcions per a gestionar persones (`persona1.h`):

```
void inputPersonaNomesNif (struct t_persona *p);
/* Efectua l'entrada del nif d'una persona, en la pantalla dissenyada per a efectuar
   L'entrada de dades de tota la persona */
```

Com a conseqüència d'aquestes noves funcions, treballarem amb unes noves versions dels paquets `generic1.h` i `persona1.h` que anomenarem `generic2.h` i `persona2.h` i que contindran les noves funcions.

El motiu de definir-ne unes noves versions és per a mantenir clara la distinció entre les versions emprades en tots els exemples desenvolupats fins el moment i les versions emprades en els exemples que desenvoluparem a partir d'ara. En realitat, aquesta distinció no seria necessària: afegiríem les noves funcions en els fitxers `generic1.h` i `persona1.h` i pordríem procedir a treballar amb ells en els nous exemples a desenvolupar i utilitzar-los també en els exemples ja desenvolupats.

Les noves funcions són:

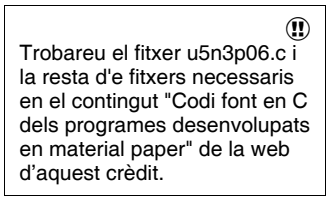
```
void inputNif (char *nif, int fila, int columna)
{
   unsigned int i;
   do
   { gotoxy(columna,fila); i = scanf ("%9s",nif); fflush (stdin); }
   while (i == 0);
   strupr (nif);
   if ( strlen(nif) < 9 )
   {
      int longitud=strlen(nif);
      int i;
      for (i=9; longitud>=0; i--,longitud--) nif[i]=nif[longitud];
      for (; i>=0; i--) nif[i]='0';
   }
}

void inputPersonaNomesNif (struct t_persona *p)
{
   pantallaPersona();
   inputNif (p->nif, 8, 40);
   gotoxy(40,8); printf ("%s",p->nif);
}

void inputPersonaSenseNif (struct t_persona *p)
{
   unsigned char i;
   long auxlong;
   double auxdouble;

   do
   { gotoxy(40,10); i = scanf ("%20s",p->cognom1); neteja_stdin(); }
   while (i == 0);

   do
   { gotoxy(40,12); i = scanf ("%20s",p->cognom2); neteja_stdin(); }
   while (i == 0);

   do
   { gotoxy(40,14); i = scanf ("%15s",p->nom); neteja_stdin(); }
   while (i == 0);

   missatge_se("Format dd-mm-aaaa amb separador '.', '-' o '/'");
   do
   { gotoxy(40,16); i = inputData (&(p->data_naix)); }
```

Trobareu els fitxers generic2.h i persona2.h amb els corresponents fitxers .c en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
   while (i == 0);

   missatge_se("Valor positiu   -   0->Si és desconegut");
   do
   { gotoxy(40,18); i = scanf ("%lf",&auxdouble); neteja_stdin(); }
   while (i == 0 || auxdouble<0);
   p->pes=auxdouble;
   /* L'anterior lectura s'ha de poder fer directament sobre &p->pes,
      però Turbo C no ho sap tractar correctament.
      En canvi, utilitzant una variable auxiliar, el tractament és correcte */

   missatge_se("Valor positiu fins a 255   -   0->Si és desconegut");
   do
   { gotoxy(40,20); i = scanf ("%lu",&auxlong); neteja_stdin(); }
   while (i == 0 || auxlong<0 || auxlong>255);
   p->alt=auxlong;
   /* En aquest cas, la utilització de la variable auxlong és per recollir el valor
      que introdueixi l'usuari, que pot ser superior a 255. Si ho féssim direc-
      tament amb la variable &p->alt, en ser un unsigned char, si l'usuari entra
      un valor superior a 255, el llenguatge C en fa el mòdul 255 i es queda amb
      el residu... No ens interessa!!! */

   netejar_missatge();
}
```

El programa que permet fer les altes al fitxer `PERS_NIF.DAT` és aquest:

```
/* u5n3p06.c */

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

#include "generic2.h"
#include "persona2.h"
#include "verscomp.h"

void main()
{
   FILE *f;
   struct t_persona p,pcerca;
   char opc;

   clrscr();
   if (existeix_fitxer ("PERS_NIF.DAT")==-1)
   {
      pregunta ("El fitxer PERS_NIF.DAT no existeix. Vol crear-lo?", &opc, "SN");
      if (opc=='N') exit (0);
   }
   pregunta ("Vol afegir persones?", &opc, "SN");
   while (opc=='S')
   {
      if ((f = fopen ("PERS_NIF.DAT","a+b")) == NULL)
      {
         missatge_ae ("Error en obrir el fitxer PERS_NIF.DAT per afegir-llegir. ");
         exit (1);
```



Trobareu el fitxer u5n3p06.c i la resta d'e fitxers necessaris en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
      }
      /* El fitxer PERS_NIF.DAT està obert en situació d'afegir i llegir registres */
      inputPersonaNomesNif (&p);
      fread (&pcerca, sizeof(struct t_persona), 1, f);
      while (!ferror(f) && !feof(f) && strcmp(pcerca.nif,p.nif))
         fread (&pcerca, sizeof(struct t_persona), 1, f);
      if (ferror(f))
      {
         missatge_ae ("Error en llegir registre del fitxer PERS_NIF.DAT. ");
         fclose (f);
         exit(1);
      }
      else if (!feof(f))
      {
         outputPersonaFor (pcerca);
         missatge_ae ("Ja existeix una persona amb aquest NIF. ");
         fclose (f);
      }
      else
      {
         inputPersonaSenseNif (&p);
         pregunta ("Vol gravar-la?", &opc, "SN");
         if (opc == 'S')
         {
            fwrite (&p, sizeof(struct t_persona), 1, f);
            if (ferror(f))
            {
               missatge_ae ("Error en gravar registre al fitxer PERS_NIF.DAT");
               fclose(f);
               exit (2);
            }
            if (fclose(f))
            {
               missatge_ae ("Error en tancar el fitxer PERS_NIF.DAT");
               missatge_ae ("Comprovi la coherència de les darreres gravacions. ");
               exit (2);
            }
         }
         else fclose(f);
      }
      pregunta ("Vol afegir més persones?", &opc, "SN");
   }
   clrscr();
}
```

Cal fer diverses observacions al programa anterior. En primer lloc, observeu el veritable tractament seqüencial, que consisteix, per cada alta que l'usuari vol efectuar, en els passos següents:

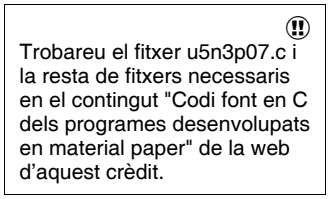
- obrir el fitxer;
- efectuar l'alta (si és factible);
- tancar el fitxer.

En realitat, el llenguatge C, mitjançant la utilització de la funció `fseek`, ens permet moure'ns per el fitxer, amunt i avall, de manera que

podríem obrir el fitxer una única vegada en iniciar el programa i tancar-lo en acabar. Els SGF seqüencials no permeten aquest tractament, sinó que únicament ens podem situar en el primer registre després d'obrir el fitxer.

D'altra banda, observeu també que en la utilització de la funció `fclose` només comprovem el valor que retorna (indicador de la correcta execució de la instrucció) en cas que hàgim efectuat una escriptura. D'aquesta manera, podem comprovar que els bytes que hagin pogut quedar en el buffer són correctament traspassats al fitxer físic.

Vegem una nova versió d'aquest programa mantenint obert el fitxer durant tota l'execució i fent servir la funció `fseek`.



Trobareu el fitxer u5n3p07.c i la resta de fitxers necessaris en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* ou5n3p07.c */

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

#include "verscomp.h"
#include "generic2.h"
#include "persona2.h"

void main()
{
   FILE *f;
   struct t_persona p, pcerca;
   char opc;

   clrscr();
   if (existeix_fitxer("PERS_NIF.DAT")==-1)
   {
      pregunta ("El fitxer PERS_NIF.DAT no existeix. Vol crear-lo?", &opc, "SN");
      if (opc=='N') exit (0);
   }
   if ((f = fopen ("PERS_NIF.DAT","a+b")) == NULL)
   {
      missatge_ae ("Error en obrir el fitxer PERS_NIF.DAT per afegir-llegir. ");
      exit (1);
   }
   /* El fitxer PERS_NIF.DAT està obert en situació d'afegir i llegir registres */
   pregunta ("Vol afegir persones?", &opc, "SN");
   while (opc=='S')
   {
      inputPersonaNomesNif (&p);
      if (fseek (f, 0L, SEEK_SET))
      {
         missatge_ae ("Error en posicionar-se a l'inici del fitxer. ");
         fclose (f);
         exit (1);
      }
      fread (&pcerca, sizeof(struct t_persona), 1, f);
```

```
      while (!ferror(f) && !feof(f) && strcmp(pcerca.nif,p.nif))
         fread (&pcerca, sizeof(struct t_persona), 1, f);
      if (ferror(f))
      {
         missatge_ae ("Error en llegir registre del fitxer PERS_NIF.DAT. ");
         fclose (f);
         exit(1);
      }
      else if (!feof(f))
      {
         outputPersonaFor (pcerca);
         missatge_ae ("Ja existeix una persona amb aquest NIF. ");
      }
      else
      {
         inputPersonaSenseNif (&p);
         pregunta ("Vol gravar-la?", &opc, "SN");
         if (opc == 'S')
         {
            fwrite (&p, sizeof(struct t_persona), 1, f);
            if (ferror(f))
            {
               missatge_ae ("Error en gravar registre al fitxer PERS_NIF.DAT");
               fclose(f);
               exit (2);
            }
         }
      }
      pregunta ("Vol afegir més persones?", &opc, "SN");
   }
   if (fclose(f))
   {
      missatge_ae ("Error en tancar el fitxer PERS_NIF.DAT");
      missatge_ae ("Comprovi la coherència de les darreres gravacions. ");
      exit (2);
   }
   clrscr();
}
```

3.6.3. Recerca d'un valor per un camp ordenat

Suposem que volem efectuar la recerca d'un valor `v` per un camp `k` ordenat de manera ascendent. Semblantment als algorismes de recerca sobre taules, podem utilitzar aquest fet per a acabar la recerca pel camp `v` quan trobem registres que tenen un valor superior al cercat.

D'altra banda sembla que no podem assegurar res sobre el caràcter identificador del camp `k`. Si no hi poden haver valors repetits, cal finalitzar la recerca en trobar-ne un. Però en cas contrari, els registres amb valor repetit estaran a continuació del primer registre trobat, per la qual cosa només cal dissenyar un procés repetitiu per a passar per tots els registres amb el mateix valor `v` una vegada trobat el primer registre amb aquest valor.

Així doncs, obtenim l'algorisme en pseudocodi següent:

```
acció recerca_ord ( v: TK) és
   var f: fitxer_seq de T;/* fitxer intern */
       x: enter; /* per a controlar l'execució de les operacions amb el fitxer */
       r: T; /* variable registre */
   fivar
   x = obrir_fs (fv, "FITXER.DAT", 'C');  /* obertura del fitxer en consulta */
   if x != 0 llavors
      error (x, "FITXER.DAT", "obrir en consulta");
      avortar_programa;
   fisi
   /* Tenim el fitxer obert */
   x = llegir_fs (f, r);
   mentre x == 0 i r.k < v fer
      x = llegir_fs (f, r);
   fimentre
   opció
      cas (x == 0 i r.k > v) o x == FI_FITXER :
         missatge ("No hi ha cap registre amb el valor cercat. ");
      cas x == 0 : /* Forçosament ha de ser r.k == v */
         tractament (r); /* procés a efectuar amb el registre */
(*)      x = llegir_fs (f, r);
(*)      mentre x == 0 i r.k == v fer
(*)         tractament (r); /* procés a efectuar amb el registre */
(*)      fimentre
(*)      si x == 0 o x == FI_FITXER llavors
(*)         missatge ("No hi ha més registres amb el valor v");
(*)      sinó
(*)         error (x, "FITXER.DAT", "llegir registre");
(*)      fisi
      altrament :
         error (x, "FITXER.DAT", "llegir registre");
   fiopció
   tancar_fs (f);
fiacció
```

Les línies marcades amb un asterisc (*) corresponen al cas en què el camp `k` no és identificador; per tant, ens interessa efectuar el tractament amb tots els registres que puguin contenir el valor `v` pel camp `k`. Si el camp `k` és identificador, només ens interessa el primer registre que contingui el valor `v`; per tant, les línies amb (*) no han de formar part de l'algorisme.

3.7. Introducció ordenada de registres

Ja coneixem l'algorisme algorisme de recerca per un camp ordenat. Hi ha, però, dues possibilitats de trobar-nos un fitxer ordenat:

- Fitxer ordenat permanentment, en el qual els processos d'altes i modificacions tenen cura de mantenir l'ordenació.

- Fitxer ordenat a causa d'haver executat prèviament un procés d'ordenació.

Ens proposem ocupar-nos de la introducció (gestió d'altes) ordenada de registres.

L'algorisme serà diferent, a més, si el camp pel qual s'ha de mantenir l'ordenació és identificador o no ho és. En cas que ho sigui, cal controlar la no-existència del valor que volem introduir. Altrament, caldrà decidir on s'ha de situar el nou registre respecte als registres existents al fitxer que contenen el valor repetit. Recordeu el concepte d'estabilitat? Potser és lògic situar el nou registre darrere de tots els que tenen el mateix valor. D'aquesta manera assegurem que davant un procés de recerca del valor, el primer registre trobat és el primer registre que va ser donat d'alta al fitxer.

Estabilitat d'un algorisme d'ordenació

Recordeu que un algorisme d'ordenació és estable si manté la posició relativa que tenien els valors repetits abans del procés d'execució

En els SGF seqüencials no és possible fer espai enmig dels registres existents. Per aquest motiu caldrà utilitzar un fitxer auxiliar i anar gravant els registres anteriors al que es vol afegir al fitxer i, posteriorment, afegir el registre que s'ha de donar d'alta seguit de tots els registres que han d'anar darrere d'aquest. Finalment, caldrà esborrar el fitxer inicial i designar el fitxer temporal amb el nom del fitxer on calia afegir el registre. (!!)

El procés descrit cal repetir-lo per cada registre que s'ha de donar d'alta. Fixeu-vos bé que per cada iteració caldrà obrir i tancar els fitxers implicats, ja que en ser fitxers seqüencials no podem tornar enrere.

Vegem, a continuació, tres algorismes en funció de les diferents variants presentades.

1) Ara dissenyarem, en primer lloc, l'algorisme en pseudocodi per al cas que el camp que ha de mantenir-se ordenat no sigui identificador i situarem tots els registres amb valor repetit en l'ordre en què s'han anat afegint (el més recent al darrere). A més del fitxer de nom `FITXER.DAT` a emplenar, ens apareix el fitxer temporal `FITXER.TMP`.

```
programa inserció_ordenada_no_identificador és
   var
      fv, fn: fitxer_seq de T; /* fitxers interns */
      rv, rn: T; /* registres */
      errorv, errorn: enter; /* control d'error en els fitxers */
      opc: caràcter;
   fivar
   pregunta ("Vol afegir registres?", opc, "SN");
   mentre opc == 'S' fer
      errorv = obrir_fs (fv, "FITXER.DAT",'C');
      si errorv != 0 llavors
```

```
   error (errorv, "FITXER.DAT", "obrir en consulta");
   avortar_programa;
fisi
errorn = obrir_fs (fn, "FITXER.TMP", 'I');
si errorn != 0 llavors
   error (errorn, "FITXER.TMP", "creació");
   tancar_fs (fv); avortar_programa;
fisi
/* L'fitxer FITXER.DAT és obert en situació de lectura. */
/* L'fitxer FITXER.TMP s'ha creat i és obert per escriptura */
llegir_T (rn);
errorv = llegir_fs (fv, rv);
mentre errorv == 0 i rv.k <= rn.k fer
   /* Fixeu-vos que la comparació <= ens situarà darrere de
      qualsevol registre amb el mateix valor pel camp k */
   errorn = escriure_fs (fn, rv);
   si errorn != 0 llavors
      error (errrorn, "FITXER.TMP", "gravar");
      tancar_fs (fv); tancar_fs (fn);
      sistema (esborrar "FITXER.TMP"); avortar_programa;
   fisi
   errorv = llegir_fs (fv, rv);
fimentre
si errorv == FI_FITXER o errorv == 0 llavors
  /* s'han llegit tots els registres anteriors al que cal inserir */
   errorn = escriure_fs (fn, rn);
   si errorn != 0 llavors
      error (errorn, "FITXER.TMP", "escriure");
      tancar_fs (fv); tancar_fs (fn);
      sistema (esborrar "FITXER.TMP"); avortar_programa;
   fisi
   /* s'ha enregistrat el nou registre - manquen els següents */
   mentre errorv == 0 fer
      errorn = escriure_fs (fn, rv);
      si errorn != 0 llavors
         error (errorn, "FITXER.TMP", "gravar");
         tancar_fs (fv); tancar_fs (fn);
         sistema (esborrar "FITXER.TMP"); avortar_programa;
      fisi
      errorv = llegir_fs (fv, rv);
   fimentre
   si errorv != FI_FITXER llavors
      error (errrov, "FITXER.DAT", "llegir");
      tancar_fs (fv); tancar_fs (fn);
      sistema (esborrar "FITXER.TMP"); avortar_programa;
   fisi
   errorn = tancar_fs (fn);
   /* comprovem error per possibles gravacions pendents */
   si errorn != 0 llavors
      error (errorn, "FITXER.TMP", "tancar");
      tancar_fs (fv); sistema(esborrar "FITXER.TMP"); avortar_programa;
   fisi
   tancar_fs (fv); sistema (esborrar "FITXER.DAT");
   sistema (renombrar "FITXER.TMP" per "FITXER.DAT");
   /* cal controlar el possible error de nova designació comunicant-ho a
      l'usuari i dient-li que el fitxer FITXER.DAT ha desaparegut i ha quedat
      el fitxer FITXER.TMP que cal tornar a anomenar */
sinó
```

```
         error (errorv, "FITXER.DAT", "llegir");
         tancar_fs (fv); tancar_fs (fn);
         sistema (esborrar "FITXER.TMP"); avortar_programa;
      fisi
      pregunta ("Vol efectuar més insercions?", opc, "SN");
   fimentre
fiprograma
```

2) Ara dissenyarem, en segon lloc, l'algorisme en pseudocodi per al cas que el camp que ha de mantenir-se ordenat sigui identificador; per tant, caldrà comprovar la no-existència d'aquest valor. De manera anàloga a l'apartat anterior, ens apareix el fitxer temporal `FITXER.TMP`.

```
programa inserció_ordenada_identificador és
   var
      fv, fn: fitxer_seq de T; /* fitxers interns */
      rv, rn: T; /* registres */
      v: TK;
      errorv, errorn: enter; /* control d'error en els fitxers */
      opc: caràcter;
   fivar
   pregunta ("Vol afegir registres?", opc, "SN");
   mentre opc == 'S' fer
      errorv = obrir_fs (fv, "FITXER.DAT",'C');
      si errorv != 0 llavors
         error (errorv, "FITXER.DAT", "obrir en consulta");
         avortar_programa;
      fisi
      errorn = obrir_fs (fn, "FITXER.TMP", 'I');
      si errorn != 0 llavors
         error (errorn, "FITXER.TMP", "creació");
         tancar_fs (fv); avortar_programa;
      fisi
      /* L'fitxer FITXER.DAT és obert en situació de lectura.
         L'fitxer FITXER.TMP s'ha creat i és obert per a escriptura */
      llegir_k (v);
      /* Observeu que només es demana a l'usuari el valor v ja que posteriorment se'n
         comprovarà la no-existència dins el fitxer per a demanar la resta de dades */
      errorv = llegir_fs (fv, rv);
      mentre errorv == 0 i rv.k < v fer
         /* A diferència de l'algorisme anterior, aquest bucle només passa pels
            registres amb valor v en el camp k */
         errorn = escriure_fs (fn, rv);
         si errorn != 0 llavors
            error (errrorn, "FITXER.TMP", "gravar");
            tancar_fs (fv); tancar_fs (fn);
            sistema (esborrar "FITXER.TMP"); avortar_programa;
         fisi
         errorv = llegir_fs (fv, rv);
      fimentre
      /* A diferència de l'algorisme anterior, en el qual només es comprovava que la
         darrera lectura s'hagués fet correctament, ara cal comprovar també la no
         existència de cap registre amb el valor que es vol afegir */
      opció
       cas errorv == FI_FITXER o (errorv == 0 i rv.k > v):
         /* cas en què podem afegir el nou registre */
```

```
         llegir_T_menys_k (v, rn);
         errorn = escriure_fs (fn, rn);
         si errorn != 0 llavors
            error (errorn, "FITXER.TMP", "escriure");
            tancar_fs (fv); tancar_fs (fn);
            sistema (esborrar "FITXER.TMP"); avortar_programa;
         fisi
         /* s'ha enregistrat el nou registre - manquen els següents */
         mentre errorv == 0 fer
            errorn = escriure_fs (fn, rv);
            si errorn != 0 llavors
               error (errorn, "FITXER.TMP", "gravar");
               tancar_fs (fv); tancar_fs (fn);
               sistema (esborrar "FITXER.TMP"); avortar_programa;
            fisi
            errorv = llegir_fs (fv, rv);
         fimentre
         si errorv != FI_FITXER llavors
            error (errrov, "FITXER.DAT", "llegir");
            tancar_fs (fv); tancar_fs (fn);
            sistema (esborrar "FITXER.TMP"); avortar_programa;
         fisi
         errorn = tancar_fs (fn);
         /* comprovem error per possibles gravacions pendents */
         si errorn != 0 llavors
            error (errorn, "FITXER.TMP", "tancar");
            tancar_fs (fv); sistema(esborrar "FITXER.TMP"); avortar_programa;
         fisi
         tancar_fs (fv); sistema (esborrar "FITXER.DAT");
         sistema (renombrar "FITXER.TMP" per "FITXER.DAT");
         /* cal controlar el possible error de nova designació comunicant-ho a
            l'usuari i dient-li que el fitxer FITXER.DAT ha desaparegut i ha quedat
            el fitxer FITXER.TMP que cal tornar a anomenar */
       cas errorv != 0 i errorv != FI_FITXER:
         error (errorv, "FITXER.DAT", "llegir");
         tancar_fs (fv); tancar_fs (fn);
         sistema (esborrar "FITXER.TMP"); avortar_programa;
       altrament:
         missatge ("Ja existeix un registre amb aquest valor.");
         visualitzar_p_T (rv);
         esperar_tecla;
         tancar_fs (fv); tancar_fs (fn);
         sistema (esborrar "FITXER.TMP");
      fiopció
      pregunta ("Vol efectuar més insercions?", opc, "SN");
   fimentre
fiprograma
```

Vegem ara una aplicació, en llenguatge C. Volem construir el fitxer `PERS_NIF.ORD` que contingui persones de manera ordenada pel camp `NIF`, que ha de ser tractat com a identificador.

Trobareu el fitxer u5n3p08.c i la resta d'e fitxers necessaris en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* u5n3p08.c */

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

#include "verscomp.h"
#include "generic2.h"
#include "persona2.h"

void main()
{
   FILE *f,*ftmp;
   struct t_persona p,pcerca;
   char opc;
   size_t size_persona = sizeof (struct t_persona);

   clrscr();
   if (existeix_fitxer ("PERS_NIF.ORD")==-1)
   {
      pregunta ("El fitxer PERS_NIF.ORD no existeix. Vol crear-lo?", &opc, "SN");
      if (opc=='N') exit (0);
   }
   pregunta ("Vol afegir persones?", &opc, "SN");
   while (opc=='S')
   {
      if ((f = fopen ("PERS_NIF.ORD","a+b")) == NULL)
      {
         missatge_ae ("Error en obrir el fitxer PERS_NIF.ORD per llegir. ");
         exit (1);
      }
      if ((ftmp = fopen ("PERS_NIF.TMP","wb")) == NULL)
      {
         missatge_ae ("Error en crear el fitxer temporal PERS_NIF.TMP. ");
         fclose (f);
         exit (1);
      }

      /* El fitxer PERS_NIF.ORD està obert en situació d'afegir i llegir.
      Només hi llegirem. S'ha obert en "afegir" per crear-lo si no existia.
      El fitxer PERS_NIF.TMP està obert per escriptura */

      inputPersonaNomesNif (&p);
      fread (&pcerca, size_persona, 1, f);
      while (!ferror(f) && !feof(f) && strcmp(pcerca.nif,p.nif)<0)
      {
         fwrite (&pcerca, size_persona, 1, ftmp);
         if (ferror(ftmp))
         {
            missatge_ae ("Error en gravar registre al fitxer temporal. ");
            fclose (f); fclose(ftmp); unlink ("PERS_NIF.TMP"); exit (1);
         }
         fread (&pcerca, size_persona, 1, f);
      }
      if (ferror(f))
      {
         missatge_ae ("Error en llegir registre del fitxer PERS_NIF.ORD. ");
```

```
         fclose (f); fclose(ftmp); unlink ("PERS_NIF.TMP"); exit (1);
      }
      else if (!feof(f) && strcmp(pcerca.nif,p.nif)==0)
      {
         outputPersonaFor (pcerca);
         missatge_ae ("Ja existeix una persona amb aquest NIF. ");
         fclose (f); fclose(ftmp); unlink ("PERS_NIF.TMP");
      }
      else
      {
         inputPersonaSenseNif (&p);
         pregunta ("Vol gravar-la?", &opc, "SN");
         if (opc == 'S')
         {
            fwrite (&p, size_persona, 1, ftmp);
            if (ferror(ftmp))
            {
               missatge_ae ("Error en gravar registre al fitxer temporal. ");
               fclose (f); fclose(ftmp); unlink ("PERS_NIF.TMP"); exit (1);
            }
            while (!ferror(f) && !feof(f))
            {
               fwrite (&pcerca, size_persona, 1, ftmp);
               if (ferror(ftmp))
               {
                  missatge_ae ("Error en gravar registre al fitxer temporal. ");
                  fclose (f); fclose(ftmp); unlink ("PERS_NIF.TMP"); exit (1);
               }
               fread (&pcerca, size_persona, 1, f);
            }
            if (ferror(f))
            {
               missatge_ae ("Error en llegir registre del fitxer PERS_NIF.ORD. ");
               fclose (f); fclose(ftmp); unlink ("PERS_NIF.TMP"); exit (1);
            }
            if (fclose(ftmp))
            {
               missatge_ae ("Error de gravació al fitxer temporal. ");
               fclose (f); fclose(ftmp); unlink ("PERS_NIF.TMP"); exit (1);
            }
            fclose (f);
            unlink ("PERS_NIF.ORD");
            if (rename ("PERS_NIF.TMP", "PERS_NIF.ORD"))
            {
               missatge_ae ("No s'ha pogut renombrar el fitxer PERS_NIF.TMP. ");
               missatge_ae ("Actuï en conseqüència!!! És el fitxer PERS_NIF.ORD. ");
               exit (1);
            }
         }
         else { fclose (f); fclose(ftmp); unlink ("PERS_NIF.TMP"); }
      }
      pregunta ("Vol afegir més persones?", &opc, "SN");
   }
   clrscr();
}
```

3) En els dos algorismes anteriors hem pogut comprovar la gran pèrdua de temps que es produeix quan cal efectuar un gran nombre d'altes en

un fitxer seqüencial, ja que no hi ha altre remei que iniciar, per cada alta que s'ha d'efectuar, un recorregut des de l'inici del fitxer, copiant registres en un fitxer temporal, fins a situar-se en el punt d'inserció per a afegir-hi el nou registre i continuar amb el procés de còpia dels registres pendents de recórrer. El procés acabarà amb l'esborrament del fitxer original i la nova designació del fitxer temporal amb el nom del fitxer original.

Fitxers seqüencials en la gestió actual

Avui en dia la gestió de fitxers no acostuma a ser seqüencial. Us imagineu l'enrenou que podria suposar efectuar una gestió d'altes de clients en fitxers seqüencials de manera que quedessin ordenats pel codi de client?

En l'actualitat, els sistemes informàtics per a gestionar grans volums d'informació haurien d'estar basats en sistemes gestors de bases de dades o, en el pitjor dels casos, en sistemes gestors de fitxers seqüencials indexats.

Per cert, us imagineu que els algorismes anteriors s'utilitzessin en un sistema multiusuari o en xarxa? Evidentment l'organització patiria un bloqueig del sistema informàtic ja que per a efectuar una alta caldria sofisticar els algorismes de manera que bloquegessin tot el fitxer mentre durés el procés. No hi podria haver accessos simultanis al fitxer.

Aquestes observacions no us han de fer pensar que el que esteu fent no us serveix per a res. Estem d'acord que molt difícilment haureu d'aplicar en la realitat un algorisme d'aquest estil. Però el coneixement d'aquests algorismes és necessari per a un bon aprenentatge de la metodologia de la programació sobre fitxers.

Davant un procés continuat d'altes, sembla lògic treballar de manera diferent. Efectuarem les altes, de manera ordenada amb els algorismes anteriors, sobre un fitxer temporal nou i, en acabar, fusionarem el fitxer real amb el fitxer creat temporalment (fent les comprovacions oportunes si el camp és identificador). És a dir, el procés consisteix en dos subprocessos encadenats:

a) Inserció ordenada en un fitxer temporal.
b) Fusió de dos fitxers ordenats en un únic fitxer ordenat.

3.8. Fusió de fitxers

A la vida real hi ha moltes ocasions en les quals necessitem utilitzar el concepte de fusió per a indicar que dues o més entitats independents, d'iguals o semblants característiques, passen a formar una única entitat. Així, en moltes ocasions hem sentit a parlar de fusions d'entitats bancàries, fusions de cadenes d'alimentació, etc.

La fusió de fitxers consisteix a construir un únic fitxer a partir de dos o més fitxers, iguals o semblants, els quals poden continuar existint.

Hi ha diferents algorismes que faciliten la fusió de fitxers. Utilitzar-ne un o un altre dependrà de les característiques dels fitxers que es vol fusionar i del fitxer resultat de la fusió. Les característiques que cal tenir en compte són aquestes:

- Ordenació dels fitxers a fusionar i del fitxer final.
- Tipus dels registres dels fitxers a fusionar i del fitxer final.
- Caràcter identificador d'un o més camps dels fitxers a fusionar i del fitxer final.

És clar que el cas més simple que ens podem trobar consisteix a ajuntar els registres dels fitxers en un sol fitxer, posant primer tots els registres d'un fitxer i, a continuació, tots els registres d'un altre, i així successivament. És evident que en aquest cas no es té en compte cap característica d'ordenació ni d'identificació. La màxima complicació es dóna si els fitxers que s'han de fusionar no tenen els registres del mateix tipus, situació davant la qual s'ha de prendre una decisió respecte al tipus de registre del fitxer final i les transformacions que s'han d'efectuar per a adaptar els registres dels fitxers originals al tipus de registre del fitxer final.

Ara presentarem l'algorisme que cal utilitzar quan s'hagin de fusionar dos fitxers, `FITXER1.DAT` i `FITXER2.DAT` de tipus `T1` i `T2` respectivament, en un fitxer `FITXER.DAT` de tipus `T`. Considerem els fitxers que es vol fusionar ordenats ascendentment pel camp `k` (comú en els tres tipus, `T1`, `T2` i `T`), el qual és identificador en els fitxers a fusionar i en el fitxer resultant. Aquest també haurà de quedar ordenat ascendentment pel camp `k`.

Considerarem que existeixen les funcions següents:

```
funció transformació1 (r1: T1) retorna T;
funció transformació2 (r2: T2) retorna T;
```

Aquestes funcions ens permeten aconseguir el registre de tipus `T` a enregistrar en el fitxer `FITXER.DAT` a partir de registres de tipus `T1` (`FITXER1.DAT`) i `T2` (`FITXER2.DAT`).

Una darrera observació. Què cal fer si detectem registres amb el mateix valor pel camp `k` en els fitxers que es vol fusionar? Cal prendre una decisió. En aquest cas, considerarem que en el fitxer final ha d'aparèixer el registre provinent del fitxer `FITXER1.DAT`. Ignorarem el registre provinent del fitxer `FITXER2.DAT`.

```
algorisme fusió_ordenada és
   var
```

```
   f1: fitxer_seq de T1; r1: T1;
   f2: fitxer_seq de T2; r2: T2;
   f: fitxer_seq de T; r: T;
   error1, error2, errorx: enter; /* control d'error en els fitxers */
fivar
error1 = obrir_fs (f1, "FITXER1.DAT",'C');
si error1 != 0 llavors
   error (error1, "FITXER1.DAT", "obrir en consulta");
   avortar_programa;
fisi
error2 = obrir_fs (f2, "FITXER2.DAT",'C');
si error2 != 0 llavors
   error (error2, "FITXER2.DAT", "obrir en consulta");
   tancar_fs (f1); avortar_programa;
fisi
errorx = obrir_fs (f, "FITXER.DAT", 'I');
si errorx != 0 llavors
   error (errorx, "FITXER.DAT", "creació");
   tancar_fs (f1); tancar_fs (f2); avortar_programa;
fisi
error1 = llegir_fs (f1, r1);
error2 = llegir_fs (f2, r2);
mentre error1 == 0 i error2 == 0 fer
   opció
      cas r1.k < r2.k : r = transformació1 (r1); error1 = llegir_fs (f1, r1);
      cas r1.k > r2.k : r = transformació2 (r2); error2 = llegir_fs (f2, r2);
      altrament: r = transformació (r1); /* ignorem el registre r2 */
                 error1 = llegir_fs (f1, r1); error2 = llegir_fs (f2, r2);
   fiopció
   errorx = escriure_fs (f, r);
   si errorx != 0 llavors
      error (errorx, "FITXER.DAT", "gravar");
      tancar_fs (f1); tancar_fs (f2); tancar_fs (f);
      sistema (esborrar "FITXER.DAT"); avortar_programa;
   fisi
fimentre
/* En aquest moment s'ha finalitzat el procés repetitiu consistent a anar comparant
   registres dels dos fitxers i enregistrant el que correspongui.
   La finalització pot ser deguda al fet que:
      S'hagi arribat a la fi d'algun (o tots dos) fitxer
      S'hagi produït un error de lectura en algun fitxer */
opció
   cas error1 != FI_FITXER i error1 != 0:
      error (error1, "FITXER1.DAT", "llegir");
      tancar_fs (f1); tancar_fs (f2); tancar_fs (f);
      sistema (esborrar "FITXER.DAT"); avortar_programa;
   cas error2 != FI_FITXER i error2 != 0:
      error (error2, "FITXER2.DAT", "llegir");
      tancar_fs (f1); tancar_fs (f2); tancar_fs (f);
      sistema (esborrar "FITXER.DAT"); avortar_programa;
   altrament:
      /* En aquest, segur que un dels dos (potser tots dos) fitxers ha arribat a la
         fi. Cal acabar d'enregistrar els registres de l'altre fitxer */
      mentre error1 == 0 fer
         r = transformació1 (r1);
         errorx = escriure_fs (f, r);
         si errorx != 0 llavors
            error (errorx, "FITXER.DAT", "gravar");
```

```
                tancar_fs (f1); tancar_fs (f2); tancar_fs (f);
                sistema (esborrar "FITXER.DAT"); avortar_programa;
             fisi
             error1 = llegir_fs (f1, r1);
          fimentre
          si error1 != FI_FITXER llavors
             error (error1, "FITXER1.DAT", "llegir");
             tancar_fs (f1); tancar_fs (f2); tancar_fs (f);
             sistema (esborrar "FITXER.DAT"); avortar_programa;
          fisi
          mentre error2 == 0 fer
             r = transformació2 (r2);
             errorx = escriure_fs (f, r);
             si errorx != 0 llavors
                error (errorx, "FITXER.DAT", "gravar");
                tancar_fs (f1); tancar_fs (f2); tancar_fs (f);
                sistema (esborrar "FITXER.DAT"); avortar_programa;
             fisi
             error2 = llegir_fs (f2, r2);
          fimentre
          si error2 != FI_FITXER llavors
             error (error2, "FITXER2.DAT", "llegir");
             tancar_fs (f1); tancar_fs (f2); tancar_fs (f);
             sistema (esborrar "FITXER.DAT"); avortar_programa;
          fisi
          /* Ja hem enregistrat tots els registres. Falta tancar el fitxer */
          errorx = tancar_fs (f);
          si errorx != 0 llavors
             error (errorx, "FITXER.DAT", "tancar");
             tancar_fs (f1); tancar_fs (f2);
             sistema (esborrar "FITXER.DAT"); avortar_programa;
          fisi
          tancar_fs (f1); tancar_fs (f2);
          missatge ("Procés finalitzat correctament.");
   fiopció
fialgorisme
```

Observem que el procés ha consistit a anar comparant registres dels dos fitxers, anar enregistrant sempre el més petit segons el camp k i passant al registre següent a través del fitxer que contenia el registre enregistrat. Quan un dels dos fitxers arriba a la fi, cal traspassar al fitxer final la resta de registres del fitxer no finalitzat.

Aquest algorisme es pot simplificar mitjançant la utilització d'un valor infinit.

> Un valor infinit per un camp ordenat és un valor que el camp pot prendre segons el tipus de dada del camp, però que no forma part del domini de valors possibles semànticament i que és més gran o més petit estrictament que tots els valors del domini segons que l'ordenació sigui ascendent o descendent respectivament.

És a dir, si definim un camp `nif` com una `cadena[10]` on els vuit primers caràcters han de ser dígits i el novè ha de ser una lletra de la `A` a la `Z`, resulta que:

- Si considerem el camp ordenat ascendentment, les cadenes `"ZZZZZZZZZ"`, `"A"`, `"A00000000"`... són valors infinits del camp, ja que són `cadena[10]`, no tenen vuit dígits seguits de lletra (no són un valor possible per al camp `nif`) i són més grans (lexicogràficament) que qualsevol `cadena[10]` que compleixi la semàntica del camp `nif`.

- Si considerem el camp ordenat descendentment, les cadenes `"000000000"`, `"0"`, `"  "` (espais en blanc)... són valors infinits del camp, ja que són `cadena[10]`, no tenen vuit dígits seguits de lletra (no són un valor possible per al camp `nif`) i són més petites (lexicogràficament) que qualsevol `cadena[10]` que compleixi la semàntica del camp `nif`.

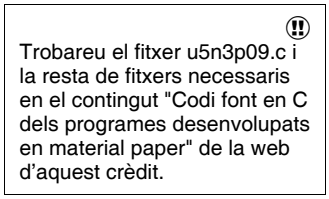
Vegem que l'algorisme anterior se simplifica bastant amb la utilització d'un valor infinit pel camp `k` com a sentinella, quan el recorregut del fitxer ha arribat a la fi. Cal anar amb compte i emprar el mateix valor infinit en tot el procés. Anomenarem v_∞ el valor infinit pel camp `k`.

```
acció llegir_f1 és
   error1 = llegir_fs (f1, r1);
   opció
      cas error1 == FI_FITXER:
         r1.k = vinf; tancar_fs (f1);
      cas error1 != 0:
         error (error1, "FITXER1.DAT", "llegir");
         tancar_fs (f1); tancar_fs (f2); tancar_fs (f);
         sistema (esborrar "FITXER.DAT"); avortar_programa;
   fiopció
fiacció

acció llegir_f2 és
   error2 = llegir_fs (f2, r2);
   opció
      cas error2 == FI_FITXER:
         r2.k = vinf; tancar_fs (f2);
      cas error2 != 0:
         error (error2, "FITXER2.DAT", "llegir");
         tancar_fs (f1); tancar_fs (f2); tancar_fs (f);
         sistema (esborrar "FITXER.DAT"); avortar_programa;
   fiopció
fiacció

/* Observeu que en cas d'arribar a la fi del fitxer, el registre llegit, que podria tenir
qualsevol valor, passa a emmagatzemar el valor infinit utilitzat en el procés. Totes les
lectures sobre els fitxers a fusionar les realitzarem mitjançant aquestes funcions.
Aquestes funcions tanquen el fitxer quan ja s'ha arribat a la fi i alliberen els recursos
utilitzats. */
```

```
algorisme fusió_ordenada_amb_valor_infinit és
   const
      vinf = v∞;
   ficonst
   var
      f1: fitxer_seq de T1; r1: T1;
      f2: fitxer_seq de T2; r2: T2;
      f: fitxer_seq de T; r: T;
      error1, error2, errorx: enter; /* control d'error en els fitxers */
   fivar

   error1 = obrir_fs (f1, "FITXER1.DAT",'C');
   si error1 != 0 llavors
      error (error1, "FITXER1.DAT", "obrir en consulta");
      avortar_programa;
   fisi
   error2 = obrir_fs (f2, "FITXER2.DAT",'C');
   si error2 != 0 llavors
      error (error2, "FITXER2.DAT", "obrir en consulta");
      tancar_fs (f1); avortar_programa;
   fisi
   errorx = obrir_fs (f, "FITXER.DAT", 'I');
   si errorx != 0 llavors
      error (errorx, "FITXER.DAT", "creació");
      tancar_fs (f1); tancar_fs (f2); avortar_programa;
   fisi
   llegir_f1;
   llegir_f2;
   mentre error1 != FI_FITXER o error2 != FI_FITXER fer
      opció
         cas r1.k < r2.k : r = transformació1 (r1); llegir_f1;
         cas r1.k > r2.k : r = transformació2 (r2); llegir_f2;
         altrament: r = transformació (r1); /* ignorem el registre r2 */
                    llegir_f1; llegir_f2;
      fiopció
      errorx = escriure_fs (f, r);
      si errorx != 0 llavors
         error (errorx, "FITXER.DAT", "gravar");
         tancar_fs (f1); tancar_fs (f2); tancar_fs (f);
         sistema (esborrar "FITXER.DAT"); avortar_programa;
      fisi
   fimentre
   /* En aquest moment s'ha acabat el procés repetitiu consistent a anar comparant
      registres dels dos fitxers i enregistrant el que correspongui.
      La finalització és deguda al fet que s'ha arribat a la fi dels dos fitxers.
      No queda cap registre pendent de procés. L'algorisme ha acabat. */
   errorx = tancar_fs (f);
   si errorx != 0 llavors
      error (errorx, "FITXER.DAT", "tancar");
      sistema (esborrar "FITXER.DAT"); avortar_programa;
   fisi
   missatge ("Procés finalitzat correctament.");
fialgorisme
```

Veurem, a continuació, la codificació en llenguatge C corresponent a l'algorisme per a fusionar dos fitxers ordenats de persones `PERS_NIF.OR1`

i `PERS_NIF.OR2`, en el qual el camp `nif` és identificador, i aconseguir el fitxer final `PERS_NIF.ORD`.


Trobareu el fitxer u5n3p09.c i la resta de fitxers necessaris en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
/* ou5n3p09.c */

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

#include "verscomp.h"
#include "generic2.h"
#include "persona2.h"

#define INFINIT "ZZZZZZZZZ"

size_t size_persona = sizeof (struct t_persona);

void llegir (FILE *f, struct t_persona *p)
{
   fread (p, size_persona, 1, f);
   if (ferror(f))
   {
      missatge_ae ("Error en llegir registre d'un dels fitxers a fusionar. ");
      fcloseall (); unlink ("PERS_NIF.ORD"); exit (1);
   }
   if (feof(f)) {  strcpy (p->nif, INFINIT); fclose (f); }
}

void main()
{
   FILE *f1,*f2,*f;
   struct t_persona r1,r2,r;
   int aux;

   if ((f1 = fopen ("PERS_NIF.OR1","rb")) == NULL)
   {
      missatge_ae ("Error en obrir el fitxer PERS_NIF.OR1 per llegir. ");
      exit (1);
   }
   if ((f2 = fopen ("PERS_NIF.OR2","rb")) == NULL)
   {
      missatge_ae ("Error en obrir el fitxer PERS_NIF.OR2 per llegir. ");
      fclose (f1); exit (1);
   }
   if ((f = fopen ("PERS_NIF.ORD","wb")) == NULL)
   {
      missatge_ae ("Error en crear el fitxer temporal PERS_NIF.ORD. ");
      fcloseall (); exit (1);
   }
   llegir (f1, &r1); llegir (f2, &r2);
   while (strcmp (r1.nif, INFINIT) || strcmp (r2.nif, INFINIT))
   {
      aux = strcmp (r1.nif, r2.nif);
      if (aux < 0) { r=r1; llegir (f1, &r1); }
      else if (aux > 0) { r=r2; llegir (f2, &r2); }
      else { r=r1; llegir (f1, &r1); llegir (f2, &r2); }
      fwrite (&r, size_persona, 1, f);
```

```
      if (ferror(f))
      {
         missatge_ae ("Error en gravar registre al fitxer PERS_NIF.ORD. ");
         fcloseall (); unlink ("PERS_NIF.ORD"); exit (1);
      }
   }
   if (fclose(f))
   {
      missatge_ae ("Error de gravació al fitxer PERS_NIF.ORD. ");
      unlink ("PERS_NIF.ORD"); exit (1);
   }
   missatge_se ("Procés finalitzat. ");
}
```

Com podeu veure, és quasi la traducció del darrer algorisme vist en pseudocodi. Hi ha, però, una diferència important. Fixeu-vos que en l'algorisme, la condició que permetia el bloc iteratiu era aquesta:

```
error1 != FI_FITXER o error2 != FI_FITXER
```

És a dir, el procés iteratiu s'anava repetint fins que tots dos fitxers havien arribat al final. Aquesta condició, en llenguatge C, seria:

```
 (!feof(f1) || !feof(f2))
```

Però no podem utilitzar la funció `feof` sobre cap dels fitxers `f1` i `f2`, ja que la funció `llegir` els tanca quan s'arriba al final del fitxer i, una vegada el fitxer és tancat, no té cap sentit aplicar-hi cap funció que no sigui la d'obertura de fitxers. Per aquest motiu, la condició emprada en llenguatge C és aquesta:

```
( strcmp(r1.nif,INFINIT) != 0 || strcmp(r2.nif,INFINIT) != 0)
```

Aquesta condició provoca el mateix efecte que la condició escrita utilitzant `feof`.

3.9. Partició d'un fitxer

Ens plantejarem ara el procés invers al de la fusió de fitxers, consistent a partir un fitxer seqüencial en dos o més fitxers en funció d'un criteri determinat. Aquest algorisme és molt simple ja que només cal fer un recorregut total pel fitxer que s'ha de partir derivant cada registre cap al fitxer que interessi.

El concepte matemàtic de partició equival a repartir els elements d'un conjunt entre diferents subconjunts, i, evidentment, un element no pot formar part de més d'un subconjunt ni quedar fora de cap subconjunt.

En informàtica, però, el concepte de partir un fitxer pot ser més ampli i abastar el fet que un element del conjunt que cal partir passi a formar part de més d'un subconjunt o quedi fora de qualsevol subconjunt. Tot dependrà dels criteris de repartiment establerts. Fins i tot és possible que l'estructura dels fitxers resultants del procés no siguin coincident entre si ni coincident amb el fitxer partit.

Vegem l'algorisme en pseudocodi consistent a partir el fitxer `FITXER.DAT` en `n` fitxers, `FITXER1.DAT`, `FITXER2.DAT`, ..., `FITXERn.DAT`, suposant que tenim les funcions següents:

```
función criteris (r: T) retorna natural;
/* Funció que, donat un registre del fitxer que cal partir, retorna un nombre natural que
indica a quin subfitxer ha d'anar a parar. Suposarem que un zero indica que no ha de
formar part de cap subfitxer. */

función transforma_x (r: T) retorna Tx;
/* Funció que, donat un registre del fitxer que cal partir, retorna el registre de tipus
Tx que caldrà inserir en el fitxer FITXER_X.DAT. */
```

L'algorisme en pseudocodi és aquest:

```
algorisme partició és
   var
      f: fitxer_seq de T; r: T;
      f1: fitxer_seq de T1; r1: T1;
      f2: fitxer_seq de T2; r2: T2;
      ..........................
      fn: fitxer_seq de Tn; rn: Tn;
      errorf: enter; errorx: taula [n] de enter; i: natural; nom : cadena [12];
   fivar
   errorf = obrir_fs (f, "FITXER.DAT",'C');
   si errorf != 0 llavors
      error (errorf, "FITXER.DAT", "obrir en consulta");
      avortar_programa;
   fisi
   error1 = obrir_fs (f1, "FITXER_1.DAT", 'I');
   si error1 != 0 llavors
      error (error1, "FITXER_1.DAT", "creació");
      tancar_fs (f); sistema (esborrar "FITXER_1.DAT"); avortar_programa;
   fisi
   error2 = obrir_fs (f2, "FITXER_2.DAT", 'I');
   si error2 != 0 llavors
      error (error2, "FITXER_2.DAT", "creació");
      tancar_fs (f); sistema (esborrar "FITXER_2.DAT");
      sistema (esborrar "FITXER_1.DAT"; avortar_programa;
   fisi
   ..................... /* Es creen els diferents fitxers... */
   errorf = llegir_fs (f, r);
   mentre errorf == 0 fer
      i = criteris (f);
      encasde criteris (f)
         ser 1: errorx[i-1]=escriure_fs (f1,transforma_1(r)); nom="FITXER_1.DAT";
         ser 2: errorx[i-1]=escriure_fs (f2,transforma_2(r)); nom="FITXER_2.DAT";
```

```
         ....................................................................
         ser n: errorx[i-1]=escriure_fs (fn,transforma_n(r)); nom="FITXER_N.DAT";
      fiencasde
      si i > 0 i errorx[i-1] != 0 llavors
         error (errorx[i-1], nom, "gravar");
         tancar_fs (f); tancar_fs (f1); tancar_fs (f2); ... ; tancar_fs (fn);
         sistema (esborrar "FITXER_1.DAT"); ... ; sistema (esborrar "FITXER_N.DAT");
         avortar_programa;
      fisi
      errorf = llegir_fs (f, r);
   fimentre
   si errorf != FI_FITXER llavors
      error (errorf, "FITXER.DAT", "llegir");
      tancar_fs (f); tancar_fs (f1); tancar_fs (f2); ... ; tancar_fs (fn);
      sistema (esborrar "FITXER_1.DAT"); ... ; sistema (esborrar "FITXER_N.DAT");
      avortar_programa;
   fisi
   tancar_fs (f);
   error1 = tancar_fs (f1);
   si error1 != 0 llavors
      error (error1, "FITXER_1.DAT", "tancar");
      tancar_fs (f2); ... ; tancar_fs (fn);
      sistema (esborrar "FITXER_1.DAT"); ... ; sistema (esborrar "FITXER_N.DAT");
      avortar_programa;
   fisi
   error2 = tancar_fs (f2);
   si error2 != 0 llavors
      error (error2, "FITXER_2.DAT", "tancar");
      tancar_fs (f3); ... ; tancar_fs (fn);
      sistema (esborrar "FITXER_1.DAT"); ... ; sistema (esborrar "FITXER_N.DAT");
      avortar_programa;
   fisi
   ..................
   /* Es tanquen els fitxers tot controlant possibles errors en gravacions pendents */
   fialgorisme
```

Com a l'exemple d'aplicació en llenguatge C, efectuarem un programa que reparteixi les persones existents al fitxer `PERSONES.DAT` en dos fitxers, `MAJORS.DAT` i `MENORS.DAT`, en funció de si són majors o menors d'edat. El programa calcularà la majoria o minoria d'edat d'acord amb la data de naixement i la data de l'ordinador en el moment d'execució.

```
/* u5n3p10.c */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>

#include "generic2.h"
#include "persona2.h"

#define MAJORIA 18

int major_edat (struct t_data d)
```

Trobareu el fitxer u5n3p10.c i la resta de fitxers necessaris en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
{
   time_t momentActual;
   struct tm mAF;  /* moment actual formatat segons els camps d'struc tm */

   time(&momentActual);
   mAF=*localtime (&momentActual);

   if ( d.any + 18 > mAF.tm_year+1900 || \
       (d.any + 18 == mAF.tm_year+1900 && d.mes > mAF.tm_mon+1) || \
       (d.any + 18 == mAF.tm_year+1900 && d.mes == mAF.tm_mon+1 && d.dia >= mAF.tm_mday))
      return 0; /* Menor d'edat */
   else return 1; /* Major d'edat */
}

void main()
{
   FILE *f, *fmaj, *fmen;
   struct t_persona p;
   size_t size_persona = sizeof (struct t_persona);

   clrscr();
   if ((f = fopen ("PERSONES.DAT","rb")) == NULL)
   {
      missatge_ae ("Error en obrir el fitxer PERSONES.DAT");
      exit (1);
   }
   if ((fmaj = fopen ("MAJORS.DAT","wb")) == NULL)
   {
      missatge_ae ("Error en crear el fitxer MAJORS.DAT");
      fcloseall(); exit (1);
   }
   if ((fmen = fopen ("MENORS.DAT","wb")) == NULL)
   {
      missatge_ae ("Error en crear el fitxer MENORS.DAT");
      fcloseall (); unlink ("MAJORS.DAT"); exit (1);
   }

   fread (&p, size_persona, 1, f);
   while (!feof(f) && !ferror(f))
   {
      if (major_edat(p.data_naix)) fwrite (&p, size_persona, 1, fmaj);
      else fwrite (&p, size_persona, 1, fmen);
      if (ferror(fmaj) || ferror(fmen))
      {
         missatge_ae ("Error en gravar registre. ");
         fcloseall (); unlink ("MAJORS.DAT"); unlink ("MENORS.DAT");
         exit (1);
      }
      fread (&p, size_persona, 1, f);
   }

   if (fclose(fmaj) || fclose(fmen))
   {
      missatge_ae ("Error en tancament de fitxer. ");
      unlink ("MAJORS.DAT"); unlink ("MENORS.DAT");
   }
   fclose (f);
}
```

3.10. Ordenació d'un fitxer

Sovint necessitem ordenar un fitxer segons uns criteris determinats. Com podem actuar?

En primer lloc hem de comentar l'existència de dues possibles ordenacions en memòria: ordenació física i ordenació lògica.

> Un fitxer és ordenat físicament per un camp k, quan un recorregut seqüencial facilita els registres ordenats pel camp k.
> Un fitxer és ordenat lògicament per un camp k, quan cada registre conté una marca al següent registre segons el valor del camp k.

La figura 6 mostra els dos exemples d'ordenació d'un fitxer per un camp `codi` de tipus numèric. El fitxer A està ordenat físicament pel camp `codi` doncs el seu recorregut seqüencial facilita els registres de forma ordenada segons el camp `codi`. En canvi, el fitxer B està ordenat lògicament pel camp `codi` via la marca que cada registre conté apuntant al següent registre segons el camp `codi`.

Figura 6. Exemple de fitxer ordenat físicament (fitxer A) i fitxer ordenat lògicament (fitxer B)

	codi	**nom**
1	3	Joan
2	5	Anna
3	8	Pep
4	22	Maria
	...	...

Fitxer A

	codi	nom	marca
1	8	Pep	3
2	3	Joan	4
3	22	Maria	
4	5	Anna	1
	...	...	

Fitxer B

2 Inici marca

En l'ordenació lògica d'un fitxer per un camp cal tenir en compte:

- Es necessita tenir informació de quin és el primer registre segons l'ordre establert. En la figura 6 s'observa que hi ha el valor 2 en "Inici marca".

- Es necessita disposar d'accés directe per posició. Si ens fixem en el fitxer B de la figura 6, observem que "Inici marca" ens diu que el primer registre seguint l'ordre és el registre número 2. I per poder-hi accedir sense passar per tots els anteriors ens cal accés directe per posició. Així mateix, en aquest registre la marca cap el següent

registre ens portaria al registre número 4, sempre i quan disposem d'accés directe per posició, i així successivament.

Com que en aquests moments estem immersos en l'estudi dels fitxers seqüencials que no permeten accés directe per posició, deixem de banda l'ordenació lògica i ens centrem en l'ordenació física. ‼

Ja coneixem alguns algorismes d'ordenació sobre taules en memòria. Com molt bé sabeu, el millor algorisme d'ordenació és l'anomenat ordenació ràpida o quicksort, que és d'ordre `n * log n`. No sabem , però, com programar aquest algorisme doncs necessitem conceptes de recursivitat que encara no coneixem, però sí que ha hem programat els típics algorismes quadràtics d'ordenació: bombolla, inserció i selecció.

Podem aplicar els algorismes d'ordenació sobre taules per a ordenar un fitxer seqüencial? La resposta és rotunda: no. Però per què no?

Recordeu qualsevol dels algorismes d'ordenació aplicats sobre taules: consistien a efectuar múltiples recorreguts per les posicions de les taules, amunt i avall, movent de lloc els continguts per tal d'aconseguir-ne l'ordenació. Aquest sistema no és possible utilitzar-lo si es treballa en fitxers seqüencials, ja que un fitxer seqüencial només permet fer un recorregut cap endavant des de l'inici, a diferència de les taules que permeten l'accés directe per posició.

Així doncs, com podem ordenar un fitxer seqüencial? Podem distingir dos tipus d'algorismes segons efectuïn o no part de l'ordenació en memòria interna (és a dir, en taules). Així, distingim:

a) Algorismes que en algun moment efectuen ordenació en memòria interna.

La situació més senzilla en la que podem emprar aquest mètode és quan el fitxer té pocs registres i aquests es poden mantenir a la vegada en una taula. En tal situació, l'algorisme consistiria en:

1) Passar tots els registres a la taula
2) Ordenar la taula en memòria interna
3) Obtenir un nou fitxer amb els registres ordenats substituint el fitxer original.

Aquesta situació tan simple és òbviament no assolible si el número de registres és elevat, de manera que no ens és possible definir una taula per a encabir-hi tots els registres. En tal situació, tenim un algorisme

alternatiu, que també utilitza una taula de capacitat màxima (però no suficient per a encabir-hi tots els registres). Els passos serien:

1) Crear un fitxer O del mateix tipus que el fitxer que s'ha ordenar, amb zero registres, el qual ha de contenir, a la fi, tots els registres ordenats.
2) Omplir una taula de capacitat màxima amb registres del fitxer que cal ordenar.
3) Aplicar un algorisme d'ordenació sobre la taula.
4) Fusionar de manera ordenada la taula amb el fitxer O ordenat fins aquell moment i generar un nou fitxer AUX que passarà a ser el fitxer O un cop finalitzada la fusió.
5) Tornar al punt 2 si en el darrer pas pel punt 2 encara han quedat registres per ordenar.

És a dir, si volem aconseguir, a partir del fitxer `FITXER.DAT` el fitxer `FITXER.ORD` ordenat pel camp `k`, l'algorisme que cal aplicar seria aquest:

```
algorisme ordenació_fitxer és
   const
      MAX = ???;
/* Valor gran de manera que hi hagi suficient memòria per a declarar la taula t */
   ficonst
   var
      f, ford: fitxer_seq de T;
      r: T;
      t: taula [MAX] de T; /* taula utilitzada per a ordenar registres en memòria */
      nt: natural; /* posicions ocupades de la taula */
      errf, errord: enter;
   fvar
   errf = obrir_fs (f, "FITXER.DAT",'C');
   si errf != 0 llavors
      error (errf, "FITXER.DAT", "obrir en consulta");
      avortar_programa;
   fisi
   errord = obrir_fs (ford, "FITXER.ORD", 'I');
   si errord != 0 llavors
      error (errord, "FITXER.ORD", "creació");
      tancar_fs (f); avortar_programa;
   fisi
   errord = tancar_fs (ford);
   si errord != 0 llavors
      error (errord, "FITXER.ORD", "tancar");
      tancar_fs (f); sistema (esborrar "FITXER.ORD"); avortar_programa;
   fisi
   /* Fitxer FITXER.ORD creat i ordenat pel camp k (en realitat buit) */
   errf = llegir_fs (f, r);
   mentre errf == 0 fer
      omplir_taula (t, nt, f, errf, r); /* procedim a omplir la taula t */
      ordenar_taula (t, nt);
      fusionar_fitxer_i_taula (t, nt);
   fimentre
   tancar_fs (f);
```

```
fialgorisme
/* Deixem com a exercici el disseny de les accions omplir_taula i ordenar_taula. Aquesta
última ha d'utilitzar un dels algorismes d'ordenació estables en cas que hi pugui haver
valors repetits pel(s) camp(s) d'ordenació */
/* Utilitzarem la tècnica del valor infinit perquè el disseny de l'algorisme de fusió
sigui més simple. Ara bé, compte, perquè no estem fusionant dos fitxers, sinó un fitxer i
una taula. */

acció llegir_ford és
   errord = llegir_fs (ford, ro);
   opció
      cas errord == FI_FITXER:
         ro.k = VALOR_INFINIT; tancar_fs (ford);
      cas errord != 0:
         error (errord, "FITXER.ORD", "llegir");
         tancar_fs (faux); sistema (esborrar "FITXER.AUX");
         tancar_fs (ford); sistema (esborrar "FITXER.ORD");
         tancar_fs (f); avortar_programa;
   fiopció
fiacció

acció fusionar_fitxer_i_taula (t: taula []de T, nt: natural) és
   var
      i: natural;
      faux: fitxer_seq de T;
      ro: T;
      erra: enter;
   fivar
   errord = obrir_fs (ford, "FITXER.ORD", 'C');
   si errord != 0 llavors
      error (errord, "FITXER.ORD", "obrir en consulta");
      tancar_fs (ford); sistema (esborrar "FITXER.ORD");
      tancar_fs (f); avortar_programa;
   fisi
   erra = obrir_fs (faux, "FITXER.AUX", 'I');
   si erra != 0 llavors
      error (erra, "FITXER.AUX", "creació");
      tancar_fs (ford); sistema (esborrar "FITXER.ORD");
      tancar_fs (f); avortar_programa;
   fisi
   llegir_ford;
   mentre  ro.k != VALOR_INFINIT o i<nt fer
      si ro.k <= t[i].k llavors
         erra = escriure_fs (faux, ro); llegir_ford;
      sinó
         erra = escriure_fs (faux, t[i]); i = i+1;
      fisi
      si erra != 0 llavors
         error (erra, "FITXER.AUX", "gravació");
         tancar_fs (faux); sistema (esborrar "FITXER.AUX");
         tancar_fs (ford); sistema (esborrar "FITXER.ORD");
         tancar_fs (f); avortar_programa;
      fisi
   fimentre
   erra = tancar_fs (faux);
   si erra != 0 llavors
      error (erra, "FITXER.AUX", "tancar");
      sistema (esborrar "FITXER.AUX");
```

```
      tancar_fs (ford); sistema (esborrar "FITXER.ORD");
      tancar_fs (f); avortar_programa;
   fisi
   tancar_fs (ford);
   sistema (esborrar "FITXER.ORD");
   sistema (tornar a anomenar "FITXER.AUX" cap "FITXER.ORD");
   /* cal comprovar la bona execució de la nova designació */
fiacció
```

Observem que l'estabilitat d'aquest algorisme d'ordenació queda garantida pels dos fets següents:

- Que l'acció `ordenar_taula` utilitzi un algorisme d'ordenació estable.

- Que en el procés iteratiu de l'acció `fusionar_fitxer_i_taula`, tenim la comparació següent:

  ```
  ro.k <= t[i].k
  ```

  Aquesta comparació garanteix que en cas d'haver-hi registres amb el mateix valor pel camp `k`, el registre que hi ha a la taula `t` és el darrer que s'enregistra al fitxer resultat de la fusió, cosa que ens interessa ja que tots els registres de la taula estaven situats, en el fitxer inicial, després de tots els registres del fitxer ordenat que es fusiona amb la taula.

El darrer algorisme presentat va ordenant els registres a trossos (segons capacitat de la taula) i per a cada tros n'efectua la fusió amb el fitxer que té ordenat fins aquell moment. Hi ha, però, una altra versió d'aquest mètode, consistent en anar generant un fitxer amb els registres de cada taula ordenada per procedir, al final del tot, a efectuar la fusió ordenada de tots els fitxers ordenats assolits. És a dir, els passos serien:

1) Omplir una taula de capacitat màxima amb registres del fitxer que cal ordenar.
2) Aplicar un algorisme d'ordenació sobre la taula.
3) Generar un fitxer ordenat amb els registres ordenats de la taula.
4) Tornar al punt 1 si en el darrer pas pel punt 1 encara han quedat registres per ordenar.
5) Fusionar de manera ordenada tots els fitxers generats en els diversos passos pel punt 3 per tal d'assolir el fitxer ordenat final.

b) Algorismes que en cap moment efectuen ordenació en memòria interna.

En aquest punt hem d'anomenar el mètode conegut com a MergeSort consistent en:

1) Dividir el fitxer F en dos fitxers F1 i F2 i ordenar per separat cadascun dels fitxers F1 i F2.
2) Procedir a una fusió ordenada dels dos fitxers F1 i F2 per a obtenir el fitxer ordenat final.

Però, com s'ordenen F1 i F2? Doncs tornant a aplicar sobre ells el mateix mètode i, per tant, necessitem, per poder-lo implementar, conceptes de recursivitat que encara no coneixem. Val a dir que aquest mètode és molt millor que els altres, doncs és d'ordre `n * log n`. mentre que els altres són d'ordre quadràtic ($\theta(n^2)$).

3.11. Actualització

En quina situació hem d'aplicar una actualització?

> Un procés d'actualització de fitxers consisteix a construir un nou fitxer mestre actualitzat a partir d'un fitxer mestre existent i d'un fitxer que emmagatzema els moviments efectuats sobre els registres del fitxer mestre.

Avui en dia aquests processos estan en decadència ja que la revolució de les comunicacions fa possible treballar en temps real des de qualsevol lloc. No obstant això, en certes ocasions, pot ser necessari efectuar processos d'actualització; per exemple quan cal actualitzar les dades d'una compta bancària a partir dels moviments efectuats en caixers automàtics que no estàven connectats a la central bancària en efectuar-se els moviments.

Cal observar que una actualització no és altra cosa que una fusió especial entre un fitxer mestre existent i un fitxer de moviments per a aconseguir un nou fitxer mestre. És especial en el sentit que no consisteix en una suma dels dos fitxers, sinó que, en funció de les característiques dels moviments (altes, baixes o modificacions), es prendran decisions sobre els registres que cal emmagatzemar en el nou fitxer mestre.

Abans de dissenyar l'algorisme, cal tenir en compte algunes consideracions sobre l'estructura i el contingut dels fitxers mestre i de moviments.

1) El fitxer de moviments és seqüencial i se suposa que els registres que emmagatzema han estat enregistrats tal com s'han anat succeint al llarg del temps.

2) El fitxer mestre ha de tenir algun camp `k` identificador, ja que si no, no se sabria sobre quin registre cal aplicar els diferents moviments. No es necessari que aquest camp sigui identificador en el fitxer de moviments, ja que sobre un mateix valor del camp `k` (únic registre del fitxer mestre) s'han pogut efectuar diferents moviments (per exemple, alta - modificació - modificació - baixa - alta - ...).

3) Quan es parla d'un procés d'actualització d'un fitxer mestre es considera que el fitxer mestre està ordenat pel camp identificador `k` i que el nou fitxer mestre ho ha de continuar estant. Podríem trobar-nos algun cas en què aquesta situació no es donés, però no acostuma a passar. Si passés, però, l'algorisme seria força més complicat i força menys eficient que el que presentarem a continuació. Més amunt hem comentat que el fitxer de moviments està ordenat temporalment. No és gens difícil aplicar-hi un procés d'ordenació estable pel camp `k`. Si hi ha diferents moviments sobre un mateix valor del camp `k`, els tindrem agrupats i ordenats segons el moment temporal en què han tingut lloc.

4) Considerarem que el registre del fitxer mestre és de tipus `T` i que el registre del fitxer de moviments és de tipus `T*`, consistent en el mateix tipus `T` més un caràcter (`'A'`, `'B'`, `'M'`) indicador del tipus de moviment (alta, baixa, modificació).

5) Així doncs, el procés d'actualització consisteix en una fusió ordenada en la qual, donat un valor pel camp `k`, cal processar-hi tots els moviments a sobre, en l'ordre temporal en què s'han produït. Cal detectar les possibles situacions d'error: alta d'un valor ja existent i baixes i modificacions de valors inexistents.

Aquestes situacions d'error cal considerar-les en el moment en què es produeixen i no pas en funció únicament dels registres emmagatzemats en el fitxer mestre que cal actualitzar. Al nou fitxer mestre hi enregistrarem, per cada valor identificador, la darrera situació vàlida. Un moviment erroni sobre un valor s'enregistra en un fitxer d'errors, i tots els moviments posteriors sobre el mateix valor també s'emmagatzemen en el fitxer d'errors, de manera que el responsable de l'actualització pugui prendre les mesures pertinents. Vegem-ne un exemple.

Suposem que en el fitxer mestre no existeix el valor `v` pel camp `k`. Suposem que en el fitxer de moviments ens trobem, per al valor `v`, la

seqüència alta - modificació - modificació - baixa - modificació - alta - modificació.

Us sembla correcta? És correcta fins que es produeix el cinquè moviment, consistent en una modificació. En efecte. D'entrada, com que en el fitxer mestre no existeix el valor, es pot efectuar el primer moviment consistent en una alta. El registre no es grava al fitxer, ja que poden quedar, com en aquest cas, altres moviments pendents. Després d'una alta, es pot efectuar el segon moviment (modificació de quelcom existent), i després d'aquest, el tercer (una altra modificació de quelcom existent), i després, el quart (una baixa de quelcom existent), i ja hi som, el cinquè moviment (modificació de quelcom inexistent ja que s'acaba de donar de baixa) provoca un error. Per tant, en el fitxer mestre no quedarà enregistrat el registre amb el valor tractat, ja que la darrera situació vàlida era que no existia perquè s'havia donat de baixa. El moviment erroni i els posteriors moviments sobre el mateix valor s'emmagatzemen en el fitxer d'errors.

El fitxer d'errors hauria de contenir registres del tipus `T*`. Evidentment, quan el responsable de l'actualització troba errors emmagatzemats, segons el tipus de moviment identifica la causa de l'error:

- Un moviment erroni de tipus `'A'` només pot ser produït per 'Alta de registre existent'.

- Un moviment erroni de tipus `'B'` només pot ser produït per 'Baixa de registre inexistent'.

- Un moviment erroni de tipus `'M'` només pot ser produït per 'Modificació de registre inexistent'.

L'algorisme d'actualització que hem estat introduint i que dissenyarem ràpidament en pseudocodi s'anomena algorisme d'actualització seqüencial amb còpia o algorisme de Feyjen-Dwyer.

```
algorisme actualització_seqüencial_amb_còpia és
   var
      fmes, fnou: fitxer_seq de T; /* fitxers mestres actual i nou */
      fmov, ferr: fitxer_seq de T*; /* fitxers de moviments i d'errors*/
      rmes: T; /* darrer registre llegit del fitxer mestre a actualitzar */
      rmov: T*; /* darrer registre llegit del fitxer de moviments */
      rnou: T; /* registre a gravar o no (segons estat final de l'aplicació dels
                  moviments) en el nou fitxer mestre */
      k: TK; /* valor del camp k pel qual s'estan tractant els moviments */
      knou: lògic; /* indica, en tot moment, si el registre rnou ha de gravar-se (o
                      no) dins el nou fitxer mestre */
      anomalia: lògic; /* indica si s'ha produït alguna anomalia en el tractament
                          dels moviments d'un determinat valor */
      errmes, errmov, errnou, errerr: enter; /* detecció d'errors de al SGF */
   fivar
```

```
   inicialitzar_procés;
   obtenir_primer_valor_a_tractar;
   mentre quedin_valors fer
      obtenir_estat_inicial_del_valor;
      mentre hi_hagi_moviments_del_valor fer
         actualitzar_estat_del_valor;
      fimentre
      actuar_segons_estat_final_del_valor;
      obtenir_següent_valor_a_tractar;
   fimentre
   finalitzar_procés;
fialgorisme

acció inicialitzar_procés és
   errmes = obrir_fs (fmes, "MESTRE", 'C');
   si errmes != 0 llavors
      error (errmes, "MESTRE", "obrir en consulta");
      avortar_programa;
   fisi
   errmov = obrir_fs (fmov, "MOVIMENT", 'C');
   si errmov != 0 llavors
      error (errmov, "MOVIMENT", "obrir en consulta");
      tancar_fs (fmes); avortar_programa;
   fisi
   errnou = obrir_fs (fnou, "MESTRE.NOU", 'I');
   si errnou != 0 llavors
      error (errnou, "MESTRE.NOU", "creació");
      tancar_fs (fmes); tancar_fs (fmov); avortar_programa;
   fisi
   errerr = obrir_fs (ferr, "MOVIMENT.ERR", 'I');
   si errerr != 0 llavors
      error (errerr, "MOVIMENT.ERR", "creació");
      tancar_fs (fmes); tancar_fs (fmov); tancar_fs (fnou);
      sistema (esborrar "MESTRE.NOU"); avortar_programa;
   fisi
   llegir_mestre;
   llegir_moviment;
fiacció

acció llegir_mestre és
   errmes = llegir_fs (fmes, rmes);
   si errmes == FI_FITXER llavors tancar_fs (fmes); rmes.k = VALOR_INFINIT;
   sinó si errmes != 0 llavors avortar_procés; fisi
   fisi
fiacció

acció llegir_moviment és
   errmov = llegir_fs (fmov, rmov);
   si errmov == FI_FITXER llavors tancar_fs (fmov); rmov.k = VALOR_INFINIT;
   sinó si errmov != 0 llavors avortar_procés; fisi
   fisi
fiacció

acció obtenir_primer_valor_a_tractar és
   k = minim (rmes.k, rmov.k);
fiacció

funció quedin_valors retorna lògic és
```

```
   retorna k != VALOR_INFINIT;
fifunció

acció obtenir_estat_inicial_del_valor és
   si k == rmes.k llavors rnou = rmes; knou = cert; llegir_mestre;
   sinó knou = fals;
   fisi
   anomalia = fals;
fiacció

funció hi_hagi_moviments_del_valor retorna lògic és
   retorna k == rmov.k
fifunció

acció actualitzar_estat_del_valor és
   opció
      cas rmov.tipus = 'A': /* moviment de tipus alta */
         si knou llavors
            anomalia = cert;
            errerr = escriure_fs (ferr, rmov);
            si errerr != 0 llavors avortar_procés; fisi
         sinó
            rnou = rmov; knou = cert;
         fisi
      cas rmov.tipus = 'B': /* moviment de tipus baixa */
         si no knou llavors
            anomalia = cert;
            errerr = escriure_fs (ferr, rmov);
            si errerr != 0 llavors avortar_procés; fisi
         sinó
            knou = fals;
         fisi
      cas rmov.tipus = 'M': /* moviment de tipus modificació */
         si no knou llavors
            anomalia = cert;
            errerr = escriure_fs (ferr, rmov);
            si errerr != 0 llavors avortar_procés; fisi
         sinó
            rnou = rmov; /* els camps que corresponguin */
         fisi
   fiopció
   llegir_moviment; /* passa al següent moviment a tractar */
   si anomalia llavors
      mentre rmov.k == k fer
         errerr = escriure_fs (ferr, rmov);
         si errerr != 0 llavors avortar_procés; fisi
         llegir_moviment;
      fimentre
   fisi
fiacció

acció actuar_segons_estat_final_del_valor és
   si knou llavors
      errnou = escriure_fs (fnou, rnou);
      si errnou != 0 llavors avortar_procés; fisi
   fisi
fiacció
```

```
acció obtenir_següent_valor_a_tractar és
   k = mínim (rmes.k, rmov.k);
fiacció

acció finalitzar_procés és
   errnou = tancar_fs (fnou);
   si errnou != 0 llavors
      error (errnou, "MESTRE.NOU", "tancar"); tancar_fs (ferr);
      sistema (esborrar "MESTRE.NOU"); sistema (esborrar "MOVIMENT.ERR");
      avortar_programa;
   fisi
   errerr = tancar_fs (ferr);
   si errerr != 0 llavors
      error (errerr, "MOVIMENT.ERR", "tancar");
      sistema (esborrar "MESTRE.NOU"); sistema (esborrar "MOVIMENT.ERR");
      avortar_programa;
   fisi
fiacció

acció avortar_procés és
   tancar_fs (fmes); tancar_fs (fmov); tancar_fs (fnou); tancar_fs (ferr);
   sistema (esborrar "MESTRE.NOU"); sistema (esborrar "MOVIMENT.ERR");
   avortar_programa;
fiacció
```

Fixeu-vos que l'algorisme presentat utilitza un VALOR_INFINIT. Es pot dissenyar l'algorisme sense fer-lo servir. Us hi animeu? Proveu-ho!