Gestió de fitxers relatius i seqüencials indexats.

6

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

Per a poder efectuar una eficaç gestió de les dades emmagatzemades en fitxers ens cal disposar de formes ràpides d'accés i això no té lloc en els ja coneguts fitxers seqüencials, on per arribar a una dada ens veiem en la necessitat de passar, abans, per les dades anteriors, ja que els fitxers seqüencials només faciliten accés seqüencial per posició.

Ens interessa, per tant, abordar altres tipologies de sistemes gestors de fitxers que facilitin altres tipus d'accés a les dades. En aquesta unitat didàctica ens proposem conèixer a fons els sistemes gestors de fitxers relatius i els sistemes gestors de fitxers seqüencials indexats.

En el nucli d'activitat "Gestió de fitxers relatius" introduïm els tipus de sistemes gestors de fitxers que possibiliten l'accés seqüencial i directe per posició i introduïm les operacions bàsiques associades relatius així com els algorismes bàsics de tractament de fitxers en els que és imprescindible l'accés directe per posició. Per tal de dur a la pràctica la gestió de fitxers relatius, utilitzarem l'SGF que aporta el llenguatge C, el qual no és exactament relatiu ni aporta les operacions teòriques pròpies d'un veritable SGF relatiu, però facilita els mecanismes perquè el programador pugui fer-ne el tractament necessari com si d'un SGF relatiu es tractés.

Dins el nucli d'activitat "Gestió de fitxers relatius" farem una introducció als fitxers calculats. En alguns llibres aquest tipus de fitxer és considerat com un SGF particular. En l'àmbit comercial no existeixen SGF calculats. Si es necessiten, cal fer-los a mida dels requeriments i, quan això succeeix, s'utilitzen fitxers relatius. Per aquest motiu, és molt lògic incloure els fitxers calculats com una aplicació dels fitxers relatius.

En el nucli d'activitat "Gestió de fitxers seqüencials indexats" abordem la gestió dels SGF seqüencials indexats després d'haver tractat en profunditat els SGF seqüencials i els SGF relatius. Iniciarem aquest nucli d'activitat amb la presentació de l'organització seqüencial indexada i de les operacions teòriques que ens acostumarà a facilitar un SGF seqüencial indexat i entrarem ràpidament a treballar amb un veritable SGF anomenat Btrieve.

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. Distingir el tipus d'accés dels fitxers relatius.
2. Identificar les operacions que faciliten els sistemes gestors de fitxers relatius.
3. Dissenyar algorismes sobre fitxers relatius per a donar resposta a les múltiples necessitats de tractament d'informació en el món real.
4. Desenvolupar programes que gestionin fitxers relatius, efectuant altes, baixes, consultes i modificacions.
5. Utilitzar, de manera correcta i eficient, la gestió directa de fitxers que facilita el llenguatge C.
6. Utilitzar, de manera correcta i eficient, els fitxers relatius per a implementar fitxers calculats.
7. Distingir el tipus d'accés dels fitxers seqüencials indexats.
8. Identificar les operacions que faciliten els sistemes gestors de fitxers seqüencials indexats.
9. Dissenyar algorismes sobre fitxers seqüencials indexats per a donar resposta a les múltiples necessitats de tractament d'informació en el món real.
10. Desenvolupar programes que gestionin fitxers seqüencials indexats, efectuant altes, baixes, consultes i modificacions.
11. Utilitzar, de manera correcta i eficient, el sistema de fitxers Btrieve.

1. Gestió de fitxers relatius

En aquests moments ja som coneixedors dels SGF seqüencials, amb les operacions teòriques que acostumen a proporcionar així com els algorismes bàsics de tractament seqüencial que s'acostuma a desenvolupar.

Ara ens ocuparà, però, el coneixement dels SGF relatius. De forma similar a l'aprenentatge dels SGF seqüencials, en primer lloc ens convé conèixer les operacions teòriques que proporcionen els SGF relatius i, posteriorment, els algorismes bàsics de gestió en fitxers relatius.

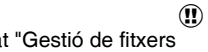
El llenguatge C no incorpora un SGF relatiu com a tal, però sí facilita operacions que permeten implementar les operacions que normalment facilita un SGF relatiu.

1.1. Operacions teòriques

Recordem la definició dels fitxers relatius:

> Un fitxer relatiu és aquell que permet l'accés seqüencial i directe per posició.

Per tant els algorismes bàsics de tractament seqüencial també són aplicables als fitxers relatius, ja que l'accés seqüencial per posició també està suportat.



A l'apartat "Gestió de fitxers seqüencials" de la unitat didàctica "Fitxers. Accés seqüencial", es presenten els algorismes bàsics de tractament seqüencial: introducció, recorregut, recerques, fusió, partició, ordenació i actualització

Per a treballar amb un SGF cal fer un estudi profund de les operacions que aporta. Presentem, en pseudocodi, les operacions que trobarem en la majoria d'SGF relatius.

Abans d'abordar les operacions que ens facilita un SGF relatiu, fem esment de la tipologia dels registres que emmagatzema. Considerarem que els registres d'un fitxer relatiu tenen, en tot moment, un determinat estat, gestionat pel mateix SGF. Tenim dos possibles estats:

- Verge: un registre és verge si no té emmagatzemat cap contingut significatiu. En realitat, tot registre emmagatzema un contingut (encara que siguin espais en blanc). El concepte de verge ens permet

decidir que el contingut emmagatzemat no correspon a una operació d'escriptura de registre en el fitxer.

- Ocupat: és el concepte contrari al de verge. Indica que el seu contingut té significat i correspon a una operació d'escriptura de registre en el fitxer.

Considerarem les operacions següents:

```
funció obrir_fr (f: fitxer_rel de T, F: cadena, mode: caràcter) retorna enter;
funció estendre_fr (f: fitxer_rel de T, num: natural) retorna enter;
funció numreg_fr (f: fitxer_rel de T, estat: caràcter) retorna enter;
funció tancar_fr (f: fitxer_rel de T) retorna enter;
funció llegir_dir_fr (f: fitxer_rel de T, r: T, pos: natural) retorna enter;
funció llegir_seq_fr (f: fitxer_rel de T, r: T) retorna enter;
funció escriure_fr (f: fitxer_rel de T, r: T, pos: natural) retorna enter;
funció modificar_fr (f: fitxer_rel de T, r: T) retorna enter;
funció esborrar_fr (f: fitxer_rel de T) retorna enter;
funció posició_fr (f: fitxer_rel de T) retorna enter;
funció posicionar_fr (f: fitxer_rel de T, pos: natural) retorna enter;
```

Observem, primer de tot, que totes les funcions tenen un argument `f` definit com a `fitxer_rel de T`. Aquest argument és el fitxer intern associat a un fitxer extern, amb el qual treballa l'SGF. Fixeu-vos que en definir el fitxer intern estem indicant el tipus relatiu 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 `fr` per indicar que són operacions de l'SGF relatiu.

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

1) Funció `obrir_fr`

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

La funció `obrir_fr` és la que efectua l'enllaç (i, per tant, reserva els canals i 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 pot incorporar o no el camí per a trobar-lo. En cas de no incorporar el camí, l'SGF cercarà el fitxer extern en el directori actual. El tercer argument `mode` permet definir el tipus d'obertura del fitxer.

Tenim tres possibilitats, presentades a la taula 1: `I` (inici), `C` (consulta), `A` (actualització).

Taula 1. Possibilitats d'actuació en un fitxer relatiu segons el mode d'obertura.

Mode	Què està en disposició de fer després d'obrir-se	Què es pot fer	Què no es pot fer
Inici	Estendre el fitxer amb registres verges.	Escriure, llegir. Estendre, modificar.	
Consulta	Llegir el primer registre existent.	Llegir.	Escriure, modificar, estendre.
Actualització	Llegir el primer registre existent. Estendre el fitxer amb nous registres verges.	Llegir, escriure. Estendre, modificar.	

El mode inici implica la creació del fitxer extern amb zero registres, 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. Podem considerar els mateixos casos que es donaven en els fitxers seqüencials. Recordem-los:

- 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 l'SGF treballarà amb la darrera versió. Es van guardant totes les versions.

L'obertura en mode inici permet executar, posteriorment, les operacions corresponents a escriure, estendre, llegir i modificar.

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

El mode actualització permet les mateixes operacions que el mode inici, amb la diferència que el mode inici crea el fitxer.

2) Funció `estendre_fr`

```
funció estendre_fr (f: fitxer_rel de T, num: natural) retorna enter;
```

Permet afegir nous registres verges al final del fitxer. És imprescindible executar aquesta instrucció després d'una creació de fitxer relatiu i abans de voler-hi escriure cap registre.

3) Funció `numreg_fr`

```
funció numreg_fr (f: fitxer_rel de T, estat: caràcter) retorna enter;
```

Aquesta funció permet saber el nombre de registres d'un determinat estat (verges, ocupats o totals) que té el fitxer. Caldrà passar, en l'argument `estat`, un dels caràcters `'V'`, `'O'` o `'T'` segons es vulgui conèixer el nombre de registres verges, el nombre de registres ocupats o el nombre total de registres.

Aquesta funció retorna un valor negatiu com a codi d'error. El retorn del valor zero indica que no hi ha cap registre amb l'estat sol·licitat.

4) Funció `tancar_fr`

```
funció tancar_fr (f: fitxer_rel de T) retorna enter;
```

La funció `tancar_fr` desfà l'enllaç entre el fitxer intern i el fitxer extern (i, per tant, allibera els recursos assignats en l'obertura). Si el fitxer era obert en els modes inici o actualització 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 i només té sentit quan hi pugui haver, en el tancament, algun enregistrament pendent.

5) Funció `llegir_dir_fr`

```
funció llegir_dir_fr (f: fitxer_rel de T, r: T, pos: natural) retorna enter;
```

La funció `llegir_dir_fr` efectua la lectura lògica del registre que es troba a la posició `pos` del fitxer. Aquesta funció ens possibilita l'accés directe per posició.

Si la posició demanada no existeix, retorna un valor diferent de zero identificat amb la constant simbòlica `NO_EXISTEIX`. També es pot produir un error quan la posició existeix, però l'estat del registre és verge. Aquest cas s'identifica amb el retorn de la constant simbòlica `ÉS_VERGE`. Sempre que es produeixi algun error, és a dir, valor retornat diferent de zero, no podrem fer cas del valor retornat per l'argument `r`.

6) Funció `llegir_seq_fr`

```
funció llegir_seq_fr (f: fitxer_rel de T, r: T) retorna enter;
```

La funció `llegir_seq_fr` efectua la lectura lògica del registre següent respecte al darrer registre a què s'ha accedit. Aquesta funció ens possibilita l'accés seqüencial per posició. És imprescindible haver accedit prèviament a algun registre amb una de les operacions `llegir_dir_fr`, `llegir_seq_fr`, `modificar_fr`, `escriure_fr`.

Suposarem que aquesta operació no té en compte els registres verges. Per tant, mai no tindrà sentit qüestionar si el valor retornat coincideix amb ÉS_VERGE. En canvi, sí que tindrà sentit qüestionar si el valor retornat coincideix amb la constant simbòlica FI_FITXER, fet que indicaria que s'ha arribat al final del fitxer.

7) Funció `escriure_fr`

```
funció escriure_fr (f: fitxer_rel de T, r: T, pos: natural) retorna enter;
```

La funció `escriure_fr` efectua l'escriptura lògica del registre `r` (segon argument) en la posició `pos` (tercer argument) del fitxer. Aquesta posició ha d'existir i ha de ser verge. Si no és així, es retornarà un codi d'error coincident amb les constants simbòliques `POSICIÓ_INEXISTENT` o `NO_VERGE` respectivament.

8) Funció `modificar_fr`

```
funció modificar_fr (f: fitxer_rel de T, r: T) retorna enter;
```

La funció `modificar_fr` efectua la modificació d'un registre existent a el fitxer que ha d'haver estat prèviament llegit (amb `llegir_dir_fr` o `llegir_seq_fr`) 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 es continua estant-hi situat al damunt, per la qual cosa es pot tornar a modificar. El registre que s´ha de modificar no pot ser verge. En cas contrari, retornaria un valor d'error equivalent a la constant simbòlica ÉS_VERGE.

9) Funció `esborrar_fr`

```
funció esborrar_fr (f: fitxer_rel de T) retorna enter;
```

La funció `esborrar_fr` retorna a verge un registre existent al fitxer que té l'estat ocupat. Igual que la funció `modificar_fr` cal estar-hi situat al damunt. El registre que s´ha d'ocupar no pot ser verge. En cas contrari, retornaria un valor d'error equivalent a la constant simbòlica ÉS_VERGE.

10) Funció `posició_fr`

```
funció posició_fr (f: fitxer_rel de T) retorna enter;
```

La funció `posició_fr` retorna la posició del darrer registre a què s'ha accedit.

11) Funció `posicionar_fr`

```
funció posicionar_fr (f: fitxer_rel de T, pos: natural) retorna enter;
```

Aquesta funció té una funcionalitat idèntica a `llegir_dir_fr`, exceptuant que no retorna cap registre, sinó que simplement se situa en el registre sol·licitat.

1.2. Accés directe per posició en el llenguatge C

El llenguatge C incorpora la possibilitat de gestionar fitxers. Sabem, però, que no té un tractament específic per a fitxers seqüencials i tampoc no té un tractament específic per a fitxers relatius. Per tant, per a treballar amb fitxers relatius mitjançant el llenguatge C, caldrà construir les funcions necessàries.

Ara bé, el llenguatge C ens aporta unes funcions per a poder-nos situar en qualsevol part del fitxer. Ja coneixem la instrucció `fseek` que utilitzàvem en l'accés seqüencial per a poder efectuar l'actualització d'un registre existent al fitxer. Recordem que la instrucció `fseek` permet tornar-se a situar en la posició inicial del registre i escriure-hi al damunt. Ens disposem ara a recordar aquesta instrucció i presentar-ne d'altres que ens facilita el llenguatge C.

Evidentment, cal tenir present que haurem de continuar utilitzant totes les instruccions ja conegudes: fopen, freopen, fclose, fcloseall, fflush, fflushall, fprintf, fscanf, fgetc, fputc, getw, putw, fgets, fputs, fread, fwrite...

1) Funció `fseek`

Aquesta funció permet situar-se en una determinada posició del fitxer a partir de la posició actual i a partir de l'inici i del final del fitxer. La seva sintaxi és aquesta:

```
#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`.

2) Funció `ftell`

Aquesta funció retorna el valor actual del punter al fitxer. La seva sintaxi és aquesta:

```
#include <stdio.h>
long ftell (FILE *f);
```

Si el fitxer és obert en mode text, el valor que retorna no ens dóna cap significat a causa de les possibles traduccions de les parelles `CR+LF`. En canvi, si el fitxer és obert en mode binari, el valor que retorna ens dóna exactament el nombre de bytes des de l'inici del fitxer.

Aquesta funció retorna el valor `-1L` en cas d'error.

3) Funció `rewind`

Posiciona el punter del fitxer en el seu inici. La seva sintaxi és aquesta:

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

La seva execució és equivalent a utilitzar la funció `fseek` per a efectuar un desplaçament de zero bytes a partir de `SEEK_SET`.

4) Funcions `fgetpos` i `fsetpos`

La seva sintaxi és aquesta:

```
#include <stdio.h>
int fgetpos (FILE *f, fpos_t *pos);
```

```
int fsetpos (FILE *f, const fpos_t *pos);
```

La funció `fgetpos` emmagatzema a la posició de memòria indicada per `pos` el valor actual del punter al fitxer. El valor emmagatzemat a `*pos` pot ser utilitzat per la funció `fsetpos`, que resitua el punter al fitxer al lloc on era en el moment d'executar la funció `fgetpos`.

El tipus de dada `fpos_t` és un tipus del llenguatge C definit en el fitxer `stdio.h`. S'utilitza per a treballar amb posicions de fitxers. Normalment, serà equivalent al tipus `long`, però en certes plataformes pot tenir un tractament diferent. Per tant, és convenient treballar amb variables d'aquest tipus en lloc de variables del tipus `long`.

1.3. Implementació de fitxers relatius

Per a gestionar un SGF relatiu no cal que tinguem en compte com s'implementen els fitxers gestionats per l'SGF. Ara bé, podem trobar-nos situacions (com en el cas del llenguatge C) en les quals es disposi de funcions d'accés als fitxers a escala byte i interessi implementar un SGF relatiu, és a dir, en les quals es pugui gestionar fitxers amb accés seqüencial i directe per posició. Com ho farem?

Hi ha tractaments diferenciats en funció de dues condicions:

a) El fitxer conté registres de longitud fixa.

Aquest és el cas més simple. Si tots els registres del fitxer tenen la mateixa longitud, és molt fàcil conèixer en quin byte comença cada registre. Per tant, utilitzant les instruccions que l'entorn ens faciliti per a situar-nos en una determinada posició (`fseek`, `rewind`, `fsetpos` en el llenguatge C, per exemple), podem implementar fàcilment l'accés directe per posició i, situats en qualsevol registre, l'accés seqüencial per posició.

b) El fitxer conté registres de longitud variable.

Aquest cas ja és bastant més complicat. Normalment, tot i que es tinguin registres de longitud variable, es considera que el fitxer manté una part fixa per a tots els registres. En aquest cas, els registres s'emmagatzemen en dues zones clarament diferenciades: la que conté les parts fixes i la que conté les parts variables. La primera té un tractament idèntic al descrit en l'apartat a). A més, els registres de la zona fixa contenen, de manera clara per a l'usuari, una direcció (número de byte) a la zona variable on està situada la corresponent part variable del registre. La figura 1 ens ho exemplifica.

Figura 1. Exemple de fitxer relatiu amb registres de longitud variable.

Zona fixa			
DNI	Nom	Sexe	@Part Variable
20500900	Teresa	D	1000
40300250	Josep	H	1325
35425900	Manel	H	1430

	Zona variable
	Hobbies
1000	Anar a comprar, gimnàs, bons restaurants
1325	Muntanya
1430	Futbol, billar, anar a fer la copa

Recordem el tractament seqüencial en els fitxers que facilita el llenguatge C: per a gravar els registres, utilitzem la funció `fwrite` indicant sempre el mateix nombre de bytes i, per tant, tots els registres tenen longitud fixa. Així doncs, podríem crear-nos la funció `llegir_dir_fr` per a accedir directament per posició a un determinat registre del fitxer. I, com no, la funció `llegir_seq_fr` per a procedir a la lectura seqüencial.

Posem-ho en pràctica per a accedir al fitxer `PERSONES.DAT` generat seqüencialment amb la gestió de fitxers que possibilita el llengutge C.

En el nucli d'activitat "Gestió de fitxers seqüencials" de la unitat didàctica "Fitxers. Accés seqüencial", el programa `u5n3p01.c` efectuava la creació i possibilitava l'emplenat del fitxer seqüencial `PERSONES.DAT`. Trobareu una còpia del fitxer `u5n3p01.c` i de la resta de fitxers necessaris en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Trobareu el fitxer `u6n1p01.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.

```
/* u6n1p01.c */

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

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

#define FI_FITXER 1
#define ERROR_LEC 2
#define ERROR_POS 3

unsigned int llegir_seq_fr (FILE *f, struct t_persona *p)
/* En cas d'error, continua en la posició on estava */
{
   fpos_t pos;
   fgetpos(f,&pos);
   fread (p, sizeof(struct t_persona), 1, f);
   if (ferror(f)) { fsetpos(f,&pos); clearerr(f); return ERROR_LEC; }
   if (feof(f)) return FI_FITXER;
   return 0;
}

unsigned int llegir_dir_fr (FILE *f, struct t_persona *p, unsigned int nreg)
/* Suposarem que la numeració dels registres comença per 1
   En cas d'error, continua en la posició on estava */
{
   fpos_t pos;
   fgetpos(f,&pos);
   if (fseek (f,(nreg-1)*sizeof(struct t_persona),SEEK_SET))
   { fsetpos(f,&pos); clearerr(f); return ERROR_POS;}
   fread (p, sizeof(struct t_persona), 1, f);
```

```
   if (ferror(f)) {fsetpos(f,&pos); clearerr(f); return ERROR_LEC;}
   return 0;
}

void main()
{
   FILE *f;
   struct t_persona p;
   unsigned int num,x;
   char opc;

   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 */
   do
   {
      gotoxy (5,23); printf ("Introdueixi número de registre a cercar:");
      missatge_se ("Introdueixi 0 per veure el registre següent.");
      do { gotoxy (46,23); x=scanf("%u",&num); neteja_stdin(); }
      while (x==0);

      if (num)
         switch (llegir_dir_fr(f,&p,num))
         {
            case ERROR_POS: missatge_ae ("No existeix una tal registre.");
                            break;
            case ERROR_LEC: missatge_ae ("Error de lectura.");
                            break;
            default: outputPersonaFor (p);
         }
      else
         switch (llegir_seq_fr(f,&p))
         {
            case ERROR_LEC: missatge_ae ("Error de lectura.");
                            break;
            case FI_FITXER: missatge_ae ("S'ha arribat al final del fitxer.");
                           break;
            default: outputPersonaFor (p);
         }
      pregunta ("Vol continuar visualitzant?", &opc, "SN");
   } while (opc=='S');
   fclose (f);
   clrscr();
}
```

Hi ha algunes observacions importants a fer. En primer lloc, observeu que hem definit unes constants simbòliques per a generalitzar el tractament d'error al llarg del programa. En segon lloc, observeu que hem implementat les funcions `llegir_seq_fr` i `llegir_dir_fr`.

És important tenir en compte que hem hagut de prendre una decisió respecte a la manera de numerar, des del punt de vista de l'usuari, els registres del fitxer per tal d'accedir-hi directament. Tot i que el llenguatge C treballa sempre a partir del byte zero, com a accés per posició sembla més natural numerar els registres a partir d'1. Així ho hem fet i, per tant, per a accedir al registre número N cal situar-se en el seu byte inicial, que es troba en la posició `(N-1)*llargada_registre`.

1.4. Fitxers relatius ordenats

Tots els algorismes introduïts en la gestió de fitxers seqüencials també són aplicables, amb poques variacions, en la gestió de fitxers relatius.

Treballem ara l'ordenació en fitxers relatius, cosa que és quasi impracticable en fitxers seqüencials. Recordeu que tenim dues possibilitats per a tenir ordenat un fitxer seqüencial: aplicar-hi un procés d'ordenació (ordenant-lo a trossos en memòria interna i fusionant-los, posteriorment, de forma ordenada) o fer-ne un manteniment ordenat (creant un nou fitxer per cada alta o modificació de manera que els registres hi quedin ordenats i tornant a anomenar el fitxer). Però els dos algorismes són molt ineficients.

1.4.1. Ordenació física i lògica de fitxers relatius

Mantenir ordenat un fitxer relatiu és més simple. Com que hi tenim accés directe per posició, un fitxer relatiu pot tractar-se de manera semblant a una taula, i en una taula sabem inserir-hi elements de forma ordenada, oi? Per tant, en un fitxer relatiu també hi hem de saber efectuar altes de forma ordenada, aprofitant els espais verges que hi hagi o desplaçant els registres que faci falta per a aconseguir espai verge per a situar el nou registre. També es pot fer servir un tractament semblant en les modificacions. El cost és, evidentment, menor que en fitxers seqüencials, però encara tenim dos problemes:

- Les E/S en fitxers són lentes, per la qual cosa, tot i que l'algorisme és senzill, el cost de temps pot ser elevat si cal efectuar molts moviments de registres (penseu en fitxers amb un gran volum d'informació).

- Només podem tenir el fitxer ordenat per un camp (o conjunt de camps). Què hem de fer si ens interessa tenir-lo ordenat segons un altre criteri? Cal aplicar, com en els fitxers seqüencials, un algorisme d'ordenació. Ja hi tornem a ser. El cost es dispara!

Els fitxers relatius, en tenir accés directe per posició, possibiliten una ordenació lògica a partir de diferents criteris, a més de l'ordenació física segons un únic criteri.

Un fitxer és ordenat físicament segons un criteri quan el seu recorregut seqüencial per posició ens proporciona els registres (exceptuant els verges) de manera ordenada segons el criteri establert.

Un fitxer és ordenat lògicament segons un criteri quan cada registre inclou, de manera clara a l'usuari, un punter a un altre registre del mateix fitxer, que és el registre següent segons el criteri establert.

Observeu que:

- En un determinat moment, un fitxer relatiu pot ser ordenat físicament segons un únic criteri i pot ser ordenat lògicament segons diferents criteris.

- L'ordenació lògica és possible en els fitxers relatius ja que hi ha la possibilitat d'accedir-hi directament per posició, de manera que, estant en un registre i coneixent la posició del registre següent, s'hi pot accedir de manera directa.

Figura 2. Exemple de fitxer relatiu ordenat físicament i lògica.

	DNI	Nom	Data naixement	Altura	Pes	Cadena nom	Cadena altura	Cadena pes
1	450	Josep	04-04-1962	170	70	3	0	3
2	870	Teresa	27-01-1958	160	55	2	3	1
3	920	Manel	25-04-1967	168	78	0	1	0
					Punters inicials per a cada cadena>>>	1	2	2

A la figura 2 podem observar que el fitxer és ordenat físicament pel camp DNI i ordenat lògicament pels camps nom, altura i pes. Veiem que cada ordenació lògica comporta l'aparició d'un encadenat de punters al registre següent.

Per cada cadena cal tenir present les consideracions següents:

- És necessari un punter inicial que ens digui quin és el primer registre del fitxer segons el criteri d'ordenació corresponent.

- El darrer registre conté el valor zero com a punter al registre següent.

Així, si volem efectuar un recorregut de manera ordenada pel camp altura, mirem el corresponent punter inicial que ens ordena efectuar un accés al registre que es troba a la posició 2 (Teresa, 160). En aquest, veiem que el valor corresponent de la cadena altura ens porta a la posició 3 (Manel, 168). I, anàlogament, aquest ens portarà a la posició 1 (Josep, 170), la qual, en tenir el valor 0 com a punter següent per la cadena altura, ens indica que ja no hi ha cap registre més. Recorreguts semblants es poden efectuar pels altres encadenaments (ordenacions lògiques).

Per a l'usuari del fitxer, els camps que contenen les cadena i els punters inicials (marcats en el dibuix amb color de fons gris molt clar) no existeixen. Els punters inicials poden emmagatzemar-se en el mateix fitxer, en uns registres de capçalera diferents dels registres que contenen les dades, o en un altre fitxer que només contingui els valors dels punters inicials.

Us imagineu el grau de complexitat en efectuar altes, baixes i modificacions en un fitxer amb ordenacions lògiques?

1.4.2. Algorismes de recerca en fitxers ordenats físicament i lògica.

Ara que ja coneixem les diferents possibilitats d'ordenació en fitxers relatius, cal passar a estudiar els mètodes de recerca que podem utilitzar per tal d'aprofitar l'ordenació existent.

Recerca en ordenació física

Suposem que tenim un fitxer relatiu de tipus `TD` ordenat físicament per un camp `tc` de tipus `TC` i que volem cercar-hi el(s) registre(s) que tinguin valor `v` pel camp `tc`. Quin mètode se us acut que podem fer servir?

Esperem que hàgiu pensat, com a mínim, en el mètode consistent a iniciar la recerca pel principi del fitxer i anar executant un recorregut seqüencial per posició fins a trobar l'element o sobrepassar el valor `v` cercat, situació en la qual podem concloure que no existeix cap registre amb el valor `v` pel camp `tc`. Ara bé, aquesta no és la resposta més eficient.

Donat que en fitxers relatius es disposa d'accés directe per posició podem anar amunt i avall per el fitxer segons ens convingui i, per tant, podem aplicar la generalització de l'algorisme de recerca dicotòmica sobre taules.

L'algorisme de recerca dicotòmica consisteix a fer successives particions, pel punt mitjà, del tros on s'ha d'efectuar la recerca, de manera que el valor només pot estar en un dels dos trossos. D'aquesta manera es pot descartar un dels dos trossos o tots dos, en el pitjor dels casos. El procés es va repetint amb el tros obtingut fins que és impossible trobar el valor cercat en algun dels dos trossos o fins que el procés finalitza, tant si s'ha trobat com si no s'ha trobat el valor cercat.

Hi ha, però, un altre mètode, idèntic en tots els passos excepte en la forma de calcular el punt de tall. Mentre que en la dicotomia el punt de tall coincideix amb el punt central del tros actual, en aquest segon mètode el punt de tall es calcula mitjançant una fórmula d'interpolació. Anomenarem aquest mètode recerca per interpolació, i el tradicional procés de recerca consistent a dividir per la meitat, recerca dicotòmica.

La recerca per interpolació també és aplicable sobre taules, de la mateixa manera que ho era la recerca dicotòmica. No és, però, gaire utilitzada, ja que no és prou coneguda. Comentem en què consisteix.

Suposem un tros (taula o fitxer amb accés directe per posició) que s'estén des de la posició inicial `IN` fins a la posició final `FI`. En tenir accés directe per posició es pot conèixer el valor emmagatzemat en les dues posicions. Suposem que aquests valors són `VIN` i `VFI` respectivament. Hem de buscar un valor `V` dins el tros. La interpolació ens diu que si els valors estan uniformement repartits entre les posicions `IN` i `FI` , el valor `V` cercat hauria de trobar-se en una posició `P` que verificaria aquesta igualtat:

$$\frac{P - IN}{FI - IN} = \frac{V - VIN}{VFI - VIN}$$

Fixeu-vos bé que tots els termes de la fórmula són coneguts excepte el valor `P` de la posició teòrica on hauria de trobar-se el valor cercat. Per tant, `P` es pot aïllar de la fórmula anterior i obtenim que:

$$P = \frac{(V - VIN) * (FI - IN)}{VFI - VIN} + IN$$

Aquesta fórmula és la que cal utilitzar per a calcular el punt de tall `P`. És a dir, s'intenta agafar com a punt de tall un punt molt proper a la posició on, per proporcionalitat de distàncies, hauria de trobar-se el valor cercat.

Fem-ne una simulació. Suposem que tenim un tros de 20 posicions [1...20] i que els valors estan bastant ben repartits. Suposem que estem cercant el valor 15. La figura 3 ens il·lustra la simulació.

Figura 3. Tros (de taula o de fitxer relatiu) de 20 posicions on es vol cercar un valor

3	7	10	12	14	16	21	23	28	30	32	37	38	40	46	49	51	54	58	60
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Per interpolació, i coneixent els valors de les posicions inicial i final del tros, s'obté 5 com a posició teòrica.

$$P = \frac{(15-3)*(20-1)}{60-3}+1 = 5$$

És a dir, obtenim que el valor cercat (15) hauria de trobar-se proper a la posició 5. Prenem aquesta posició com a punt de tall i comprovem en quin dels dos trossos ([1...5] o [6...20]) pot estar el valor cercat. Com que 15 és més gran que el valor existent a la posició 5 i és més petit que el valor existent a la posició 6, es dedueix que el valor cercat no existeix. El procés iteratiu ha consistit en un únic pas.

Si haguéssim utilitzat la recerca dicotòmica, el punt de tall hauria estat la posició 10 i hauríem repetit el procés per al tros [1...10]. En tornar a calcular el punt de tall, hauria estat la posició 5, per a la qual ja es dedueix la no-existència del valor cercat. Però hem hagut de fer dos passos.

Suposem ara que cerquem el valor 58 per interpolació. Aplicant la fórmula obtenim 19 (arrodonint a 0 decimals).

$$P = \frac{(58-3)*(20-1)}{60-3}+1 \approx 19$$

El procés ens obliga a decidir entre el tros [1...19] o [20...20]. Prenem el primer. Tornem a aplicar la fórmula i obtenim 19.

El procés ens obliga a decidir entre el tros [1...18] i [19...19]. Prenem el segon i arribem a la conclusió que hem trobat l'element cercat. Només hem repetit el procés dues vegades.

Utilitzant la recerca dicotòmica, hauríem tallat per la posició 10 i hauríem pres el tros [11...20]. Posteriorment, hauríem tallat per la posició 15 i ens hauríem quedat el tros [16...20]. En tercer lloc, tallaríem per la posició 18 i ens quedaríem el tros [19...20]. En quart lloc, tallaríem per la posició 19 i ens quedaríem el tros [19...19] i deduiríem l'existència del valor cercat. Hauríem utilitzat quatre passos.

En els exemples anteriors hem pogut comprovar el millor comportament de la interpolació respecte a la dicotomia per a calcular el punt de tall. Per què no comproveu com es comporta en un cas en què els valors emmagatzemats estiguin molt mal repartits?

En general, podrem utilitzar la recerca per interpolació si:

- El conjunt ordenat de valors per als quals s'ha d'efectuar la recerca està uniformement repartit. El grau d'eficiència del càlcul del punt de tall està en concordança amb el repartiment uniforme dels valors del conjunt.

- El conjunt de valors per als quals s'ha d'efectuar la recerca és de tipus numèric. Observeu que la fórmula els utilitza per a efectuar operacions aritmètiques. I si fossin noms en lloc de nombres? No hi ha problema si disposeu d'algun algorisme que transformi els noms en nombres de manera que el conjunt numèric obtingut continuï estant ordenat i presenti una uniformitat en el repartiment de valors transformats equivalent a la uniformitat existent en el repartiment dels valors reals.

La interpolació ens apropa molt ràpidament a un punt de tall òptim si el conjunt està ben repartit. La dicotomia triga més, però és més segura ja que no depèn de la uniformitat de la distribució. Es pot aplicar la recerca mixta, consistent a utilitzar els mètodes de dicotomia i interpolació de manera alternativa en el càlcul de cada valor de tall.

A continuació presentem, en pseudocodi, una funció que ens efectua la recerca en qualsevol dels tres casos (dicotomia, interpolació, mixta) sobre fitxers relatius. Suposem que hem de cercar el valor `v` pel camp `tc` en un tros de fitxer que s'estén des de la posició `inici` fins a la posició `final` i per al qual suposem que no conté cap registre verge. Intenteu millorar l'algorisme de manera que sàpiga tractar registres verges.

```
funció recerca_ordenada (f: fitxer_rel; ini, fin: natural; m: caràcter; var r: TD;
                         v: TK) retorna enter és
/* Arguments: f: fitxer relatiu on cal efectuar la recerca; se suposa obert
              ini, fin: posicions extremes del tros de fitxer on cal cercar
              m: mètode a emprar, amb valors possibles 'D', 'I' i 'M'
              r: variable que recollirà el registre cercat (si es troba)
   Retorna: La posició en què es troba el valor cercat
            Retorna -1 si no s'ha trobat
            Retorna -2 si hi ha paràmetres incorrectes
*/
   var tall, parell: natural; rini, rfin: TD; possible: lògic; fivar
   si (m!='D' i m!='I' i m!='M') o (ini>fin) llavors retorna -2 fisi
   llegir_dir_fr (f, rini, ini); llegir_dir_fr (f, rfin, fin);
   possible = rini.tc <= v i v <= rfin.tc; parell = 0;
```

```
   mentre possible i ini < fin fer
      opció
         cas m == 'D' o (m == 'M' i parell == 0): /* dicotomia */
             tall = (ini + fin ) div 2;
             parell = 1;
         cas m == 'I' o (m == 'M' i parell == 1): /* interpolació */
             tall = (v - rini.tc) * (fin - ini) / (rfin.tc - rini.tc) + ini;
             parell = 0;
      fiopció
      llegir_dir_fr (f, r, tall);
      si v <= r.tc llavors fin = tall; rfin = r;
      sinó llegir_dir_fr (f, r, tall+1);
           si v >= r.tc llavors ini = tall+1; rini = r;
           sinó possible = fals;
           fisi
      fisi
   fimentre
   si rini.tc == v llavors r=rini; retorna ini; fisi
   retorna -1;
fiprograma
```

En el programa anterior cal afegir el corresponent tractament d'error en efectuar les instruccions de lectura al fitxer.

Recerca en ordenació lògica

L'ordenació lògica que coneixem, consistent en mantenir una cadena de punters a la posició següent segons el criteri de classificació, és clarament millorable amb l'ordenació lògica a salts.

> Un fitxer està ordenat lògicament a salts segons un criteri quan inclou dues cadenes de punters: una primera que permet passar de cada registre al seu immediat següent i una segona que permet passar d'un registre al registre que es troba s (salt) posicions (segons l'ordre) més enllà.

La figura 4 ens exemplifica un fitxer relatiu ordenat lògicament a salts pel camp nom. S'observen dues cadenes de punters. La primera, anomenada cadena simple nom, consisteix en una cadena simple que per cada registre proporciona el registre següent segons l'ordre lexicogràfic. La segona, anomenada cadena salts nom, consisteix en una cadena que permet accedir als registres efectuant salts de 3 en 3.

Cada cadena té un punter inicial, que ens adreça al primer registre. Així, per a la cadena simple, el primer registre correspon al que està en la posició 10, de nom Benet. Per a la cadena a salts de 3 en 3, el primer

registre (una vegada efectuat el primer salt de tres registres) correspon al que està en la posició 8, de nom Elvira.

Figura 4. Exemple de fitxer relatiu ordenat lògicament a salts pel camp "nom"

	DNI	Nom	Data naixement	Altura	Pes	Cadena simple nom	Cadena salts nom
1	450	Oriol	09-12-1980	169	65	6	0
2	870	Carlota	19-03-1936	165	70	8	-1
3	920	Manel	25-04-1967	168	74	7	-1
4	225	Guifré	07-02-1995	100	16	5	-1
5	342	Joan	09-12-1960	175	85	9	-1
6	425	Regina	05-12-1992	105	18	0	-1
7	972	Manela	24-11-1932	155	65	1	-1
8	782	Elvira	09-01-1965	165	60	4	9
9	645	Joan	07-01-1924	174	75	3	1
10	123	Benet	03-08-1931	158	50	2	-1

Inici simple	Inici salts
10	8

En un fitxer ordenat lògicament a salts, l'algorisme de recerca ordenada consta dels passos següents:

1) Resseguir els registres per la cadena a salts fins a trobar o sobrepassar el valor cercat o fins a arribar a la fi de la cadena.

2) Si s'ha trobat el valor cercat i no n'hi ha de repetits (fet que no succeeix amb el camp nom), es conclou que s'ha trobat el registre amb el valor cercat. Si s'ha trobat el valor cercat i n'hi ha de repetits, el registre trobat no cal que sigui el primer registre amb el valor cercat. Si simplement es volia trobar un registre, el procés finalitza. Ara bé, si es volia cercar el primer registre que contingués el valor cal passar al punt 3), de la mateixa manera que cal passar-hi si en el punt 1) s'havia sobrepassat el valor cercat o s'havia arribat a la fi dels registres segons la cadena a salts.

3) Cal "desfer" el darrer salt. És a dir, a partir del registre des del qual s'ha efectuat el darrer salt, es parteix cap al registre indicat per la cadena simple. A partir d'aquests moments, s'efectua una recerca seqüencial seguint únicament la cadena simple fins a trobar o sobrepassar el valor cercat o fins a arribar a la fi de la cadena. Si es troba, segur que aquest és el primer registre amb el valor cercat. Si se sobrepassa o s'arriba a la fi de la cadena simple, cal concloure que no hi ha cap registre amb el valor cercat.

Cal observar, en l'exemple, com la cadena a salts només té valors per uns determinats registres. Els registres "saltats" en la cadena a salts tenen, com a punter al registre següent, un valor que indica que no hi ha punter. Hem utilitzat el valor -1.

Exemplifiquem les possibles recerques sobre el fitxer exemplificat a la figura 4:

a) Recerca del nom Xavier

La cadena a salts ens indica que hem de començar pel registre 8. El llegim i hi trobem el nom Elvira, anterior a Xavier. En aquest registre el punter per la cadena de salts ens porta al registre 9. El llegim i hi trobem el nom Joan anterior a Xavier. D'aquí, per la cadena de salts, anem al registre 1, on trobem el nom Oriol, anterior a Xavier. En aquesta posició trobem que la cadena de salts conté un 0 (zero), marca que indica que s'ha arribat al final dels registres per la cadena de salts. Hem de continuar, per tant, a partir d'aquest registre, per la cadena simple. Arribarem al registre 6, on trobem Regina, i després trobem el 0 (zero) com a registre següent per la cadena simple. Conclusió: no hi ha cap Xavier al fitxer.

b) Recerca del nom Anna

El punter inicial de la cadena de salts ens porta al registre 8, on trobem Elvira, que és posterior al valor cercat. Per tant, hem de partir de la cadena simple, que ens porta al registre 10, on trobem Benet, que és posterior al valor cercat. Conclusió: no hi ha cap Anna a el fitxer.

c) Recerca del nom Joan

El punter inicial de la cadena de salts ens porta al registre 8, on trobem Elvira, anterior a Joan. El punter de la cadena de salts ens porta al registre 9, on trobem Joan, que coincideix amb el valor cercat. Com que el camp nom no és identificador, hi pot haver algun registre anterior amb nom Joan. Per tant, tornem a partir del registre des del qual havíem efectuat el darrer salt (8, Elvira) i seguim per la cadena simple. Inicialment anem al registre 4, on trobem Guifré, anterior a Joan. D'aquí anem al registre 5, que conté Joan. Hem trobat el primer registre que conté el valor cercat.

Per acabar, tingueu en compte un parell de comentaris sobre l'ordenació lògica a salts:

- Si el fitxer té de l'ordre de N registres, el salt òptim per a construir la cadena a salts és $\sqrt{N}$.

- La cadena a salts degenera ràpidament si el fitxer es manté constant (altes, baixes, modificacions). Caldrà, en aquest cas, refer periòdicament la cadena de salts.

Comparativa d'eficiència

La taula 2 ens permet comparar el nombre d'operacions necessàries en els diversos mètodes de recerca ordenada.

Taula 2. Ordre d'operacions a efectuar en els diferents mètodes de recerca ordenada.

Mètode	Mitjà	Màxim	Suposant N = 100.000	
			Mitjà	Màxim
Dicotomies	$\log_2 N$	$\log_2 N + 1$	16	17
Interpolació	$\log_2(\log_2 N)$	N	4	100.000
Mixta	$2\log_2(\log_2 N)$	$2 \log_2 N$	8	32
A salts	$\sqrt{N}$	$2 \sqrt{N}$	316	632

S'observa clarament que el mètode d'interpolació és el millor, quan s'aplica en la situació òptima (distribució de valors uniforme), mentre que és molt dolent quan la situació en la qual s'aplica és totalment desencertada.

A partir dels valors anteriors potser deduiríem que està bé utilitzar la recerca dicotòmica, ja que és molt estable. El raonament és força correcte. A més, és possible que penseu a rebutjar el mètode de recerca a salts, ja que és força dolent en comparació amb la recerca dicotòmica. Però oblideu un factor molt important: els tres primers mètodes impliquen tenir el conjunt de valors (taula o fitxer relatiu) ordenat físicament pel camp que conté el valor que s'ha de cercar. Quin cost té aconseguir això? En canvi, la recerca a salts necessita mantenir el conjunt ordenat lògicament. Això també té un cost, però podem tenir, en un mateix conjunt, diferents ordenacions lògiques, mentre que d'ordenació física només n'hi pot haver una.

1.5. Fitxers calculats

Sabem que de SGF comercials n'hi ha principalment 3 tipus: seqüencials, relatius i seqüencials indexats. Hi ha, però, 2 tipus més de fitxers importants: els calculats i els textuals.

Ens proposem, ara, endinsar-nos en el coneixement dels fitxers calculats, també anomenats aleatoris o hashing, els quals no estan comercialitzats i que s'implementen amb la utilització de fitxers relatius. Per aquest motiu,

sovint se'ls considera un cas particular de fitxers relatius, i per tant, s'acostumen a incloure en els capítols dedicats a l'estudi dels fitxers relatius.

> Els fitxers calculats permeten l'accés seqüencial i directe per posició (ja que es basen en fitxers relatius) i l'accés directe per valor d'una única via d'accés.

En els fitxers relatius es pot accedir als registres a través de la posició en què s'han inserit. No tenim cap informació que ens permeti intuir on es troba un determinat registre. Com a màxim, si el fitxer està ordenat físicament, segons sigui la uniformitat de la distribució de valors, podrem fer interpolació. També podrem intuir si hem d'anar a buscar pel principi o pel final segons que el valor sigui dels primers o dels darrers. Tot això és, però, molt pobre per a poder cercar registres pel valor d'un camp o un conjunt de camps.

La situació que presentem no és nova. Ja es va presentar en la dècada dels setanta quan es treballava amb fitxers relatius. I els informàtics de l'època se les van haver d'enginyar per trobar alguna solució efectiva.

Imaginem-nos que volem emmagatzemar els alumnes de l'escola de manera que puguem cercar-los més o menys ràpidament per algun valor que els identifiqui. D'entrada podem pensar en dues possibilitats:

a) Suposant que cada alumne tingui un número d'alumne o de matrícula que l'identifiqui dins l'escola, podríem tenir el registre corresponent a l'alumne de número N situat a la posició N del fitxer d'alumnes. D'aquesta manera tenim accés directe per un valor de número de matrícula, ja que aquest número coincideix amb la posició física on es troba l'alumne.

Aquesta opció té un problema. Si els números de matrícula es van succeint al llarg dels cursos, podem trobar-nos que els alumnes actuals de l'escola tinguin números de matrícula entre el 3001 i el 3500. Hauríem de tenir un fitxer amb 3.500 posicions on les 3.000 primeres estarien verges. Així perdem molt espai.

b) I si considerem el DNI de l'alumne com la posició del fitxer on podem tenir situat l'alumne?

En aquest segon cas també tenim un greu problema d'espai. Si suposem que el valor del màxim DNI és 99.999.999, necessitarem que el nostre fitxer tingui 100 milions de registres. Molt pocs estaran ocupats, oi?

Aprofundint en les dues idees veiem que el problema és que el rang de valors possibles és molt ampli enfront del conjunt de valors reals. ¿I si aconseguíssim reduir el gran espai de valors possibles a un petit espai d'adreces, en el qual fiquéssim els registres de manera que tinguéssim una manera ràpida d'accedir-hi?

En la primera de les possibilitats anteriors, podríem pensar a tenir un fitxer de 500 posicions de manera que l'alumne amb matrícula N el ficaríem a la posició N-3000. Si en les insercions utilitzem sempre la mateixa norma, aquesta ens servirà també per a trobar, posteriorment, els registres.

> Un fitxer calculat és un fitxer relatiu en el qual cada registre té assignat, en funció del valor d'un camp o un conjunt de camps, una posició dins el fitxer.

El terme calculat prové del fet que la posició que ha d'ocupar cada registre es calcula a partir del valor del camp o conjunt de camps, cosa que fa deduir, per tant, l'existència d'una funció de càlcul, responsable de l'assignació de la posició segons el valor del camp o conjunt de camps. Ens cal, doncs, disposar de diferents funcions de càlcul i de mètodes per a gestionar, de manera efectiva i segura, els fitxers calculats. a

Utilitzarem el terme clau per a referir-nos al valor del camp o conjunt de camps pel qual s'ha de calcular la posició o adreça dins el fitxer relatiu. A més, suposarem que el camp o conjunt de camps és identificador, és a dir, que no hi poden haver valors repetits. a

1.5.1. Transformació clau-adreça

Per a calcular l'adreça que correspon a una clau disposem de diferents mètodes:

- transformacions directes,
- taules de consulta,
- transformacions aleatòries.

a) Transformacions directes

Aquest mètode, que es pot fer servir quan les claus són nombres consecutius o hi ha molt pocs forats, consisteix en assignar la posició 1 del fitxer al menor valor possible de la clau, la posició 2 al següent valor, i així successivament.

Com a exemple d'aquest tipus de transformació, tenim el fitxer d'alumnes d'una escola en el que utilitzem el número de matrícula de cada alumne com el valor per a assignar directament el número de registre en el que ha de residir cada alumne. Així, si les matrícules van numerades des del número 3000 fins el número 3500, podem fer l'assignació següent: l'alumne amb número de matrícula 3000, en el registre que ocupa la posició 1; l'alumne amb el número de matrícula 3001, a la posició 2; i així successivament.

Per tant, la funció de càlcul es pot representar amb aquesta senzilla fórmula matemàtica:

```
posició = clau – menor_clau_possible + 1
```

És a dir, en notació matemàtica tenim que:

```
f(clau) = clau + K
```

on `K` és una constant amb valor igual a `1 – menor_clau_possible`.

- Avantatges d'aquest mètode

– No cal emmagatzemar el valor clau al fitxer, ja que és calculable a partir de la posició on es troba el registre. Per tant, s'estalvia espai.

– El fitxer queda ordenat físicament per la clau.

– No hi ha sinònims (registres que els pertoca la mateixa adreça) ni excedents (registres sinònims que no caben a l'adreça que els pertoca).

Els conceptes "sinònim" i "excedent" es presenten a l'apartat "Sinònims o col·lisions"

- Inconvenients d'aquest mètode

– Es poden produir una sèrie de forats al fitxer com a conseqüència, d'una banda, de l'existència de valors de claus per les que no hi pot haver registres i, de l'altra, de la fusió de fitxers d'aquest tipus amb la utilització d'una clau prèvia a les claus existents per a identificar la provinença de cada registre.

  En definitiva, a causa dels forats que es puguin produir, aquest mètode és molt poc utilitzat.

b) Taules de consulta

Aquest mètode consisteix a disposar d'una taula que associa a cada clau la posició del fitxer, tal com es veu a la taula 3.

Taula 3. Taula de consulta per a una funció de hash

Clau	Posició
345	4
423	6
822	2
...	...

Per a cercar un registre al fitxer, primer cal identificar-ne la clau a la taula, on trobarem la posició del fitxer a la que hem d'anar a parar. Aquest mètode és com un tipus d'indexació.

A l'apartat "Gestió de fitxers seqüencial-indexats" coneixerem les possibilitats d'indexació.

Com es manté la taula? Hi ha diverses possibilitats:

- Mantenir la taula en memòria mentre es treballa amb el fitxer calculat guardant-la en un fitxer auxiliar quan es tanca el fitxer calculat i tornant-la a carregar en memòria en obrir el fitxer calculat.

  Aquesta gestió ens pot produir problemes de memòria si el fitxer calculat té un gran nombre de registres.

- Mantenir la taula en memòria que es crea en obrir el fitxer calculat i es destrueix en tancar el fitxer calculat sense tenir el suport d'un fitxer auxiliar.

  Aquesta gestió també ens pot produir problemes de memòria com en el cas anterior i, a més, tenim la necessitat d'efectuar un recorregut per tot el fitxer calculat en obrir-lo per tal de construir la taula.

- Mantenir la taula en un fitxer auxiliar que complementa el fitxer calculat.

  Aquesta gestió té problemes d'eficiència ja que el fitxer auxiliar s'ha de mantenir ordenat per la clau d'accés al fitxer calculat, i ja coneixeu els costos de mantenir ordenat un fitxer, oi?

c) Transformacions aleatòries

Les transformacions aleatòries són les més utilitzades per a la gestió de fitxers calculats. Per aquest motiu, els fitxers calculats també s'anomenen fitxers aleatoris. El terme aleatori neix arran de la sensació de càlcul atzarós (tal com podrem comprovar) per a obtenir la posició a partir de la clau. En realitat no pot ser un càlcul aleatori, ja que donada una clau

el càlcul ha de donar sempre la mateixa posició per tal de poder cercar el registre corresponent.

Les transformacions aleatòries consisteixen a esmicolar o trinxar la clau (de seguida ho comprovarem) per a obtenir una posició dins el rang de posicions permeses, rang limitat al nombre natural que representa la quantitat de registres del fitxer relatiu. El terme anglès corresponent a esmicolar és hash, per això els fitxers calculats també són coneguts com a fitxers hashing i la funció de càlcul és coneguda com a funció hash.

Els mètodes hashing consten, en general, de quatre passos:

1) Selecció de dígits o caràcters amb rellevància, dins la clau, en cas que les claus tinguin una part repetitiva.

Per exemple, quan tots els telèfons d'una província començaven amb els mateixos dígits, una funció de hashing sobre claus telefòniques podria començar seleccionant els dígits amb rellevància, és a dir, eliminant els dígits corresponents a l'identificador de la província.

2) Transformació de la clau alfanumèrica a numèrica.

Per a aconseguir-ho podem inventar-nos qualsevol mètode. Per exemple, podem prendre la suma de les posicions, dins la taula ASCII, de tots els caràcters que formen la clau.

3) Aplicació d'un(s) algorisme(s) de transformació per tal d'obtenir un conjunt de posicions proporcional a la quantitat de registres que ha de tenir el fitxer relatiu, de manera que dins el conjunt de posicions, les claus s'han de distribuir tan homogèniament com sigui possible.

4) Multiplicació dels valors resultants de la transformació per un factor, amb la finalitat que totes les claus estiguin dins el rang de posicions del fitxer.

En definitiva, si es vol treballar amb un fitxer relatiu de N posicions, cal assegurar que:

- La funció hash proporcionarà sempre un valor entre 1 i N.

- Tots els valors entre 1 i N han de ser assolits per la funció hash.

Així, si per a un fitxer de 100 registres es considerés la funció de hash H(clau) = 100, tindríem que és una funció que sempre dóna com a resultat 100 i, per tant, verifica que el seu resultat sempre estigui entre 1 i 100, però, per altra banda, cap registre no podria accedir mai als valors existents entre 1 i 99, ja que aquesta funció hash mai no

proporciona aquest resultat. Evidentment, la funció presentada no serveix.

Vegem, a continuació, alguns dels algorismes de transformació a què fa referència el segon pas (transformació de la clau alfanumèrica a numèrica). A vegades només se n'aplica un i altres vegades se n'apliquen diversos consecutivament.

- Si la clau és curta pel rang de valors desitjable, podem aplicar el mètode del centre al quadrat.

  Consisteix a elevar la clau al quadrat i prendre'n la part central. Així, per a obtenir una clau de cinc xifres a partir de valors de quatre xifres com el 5864, podem calcular-ne el quadrat 34(38649)6 i quedar-nos la part central, 38649.

- Si la clau és llarga pel rang de valors desitjable, podem aplicar el mètode de doblegar o plegar.

  Consisteix a doblegar els extrems de la clau sobre si mateixos (com si dobleguéssim un full de paper). Els dos nombres formats se sumen. Així, per a obtenir una clau de 5 o 6 xifres a partir d'una clau de 10 xifres com 1234567890, podem:

  – Doblar el nombre per l'extrem esquerre 1 fins al 4 i per l'extrem dret 0 fins al 5. Obtindrem els nombres 21098 (superior) i 34567 (inferior).

  – Sumar els dos valors. Obtindrem 55665.

- Un altre mètode utilitzat quan la clau és massa llarga i es vol escurçar, és el mètode de desplaçar.

  Consisteix a desplaçar els extrems de la clau cap endins dels nombres (com si fossin portes corredisses) fins a obtenir dos nombres de la mateixa longitud que el rang de valors desitjats. Així, per exemple, en el cas de l'apartat anterior, per a obtenir un valor de 5 xifres tindríem 12345 + 67890, és a dir, 80235.

- Un altre mètode és el de la divisió, consistent a prendre el residu de la divisió sencera de la clau per un nombre primer, proper al nombre de registres del fitxer. Són aconsellables els valors primers superiors a 19.

  Aquest és el millor mètode després d'haver aplicat un plegament o desplaçament o d'haver utilitzat el mètode del centre al quadrat

segons necessitem disminuir o augmentar la quantitat de dígits de la clau.

- Un cinquè mètode és el de la conversió d'arrel, consistent a canviar l'arrel del nombre i suprimir els dígits de major ordre en la sèrie de dígits resultants.

  Per exemple, treballant amb l'arrel 13, la clau 12345 es converteix a:

  $$1 * 13^4 + 2 * 13^3 + 3 * 13^2 + 4 * 13^1 + 5 = 33519$$

  Suprimint els valors superiors (33) obtenim 519 com a resultat final.

- L'últim mètode és el de la divisió polinòmica. Cada dígit és pres com a coeficient d'un polinomi, el qual es divideix amb un polinomi fix. Els coeficients del residu s'utilitzen com a resultat de la transformació.

Hashings **dinàmics**

Observeu que tots els mètodes presentats tenen en compte la quantitat de registres que pot emmagatzemar el fitxer relatiu, ja que el conjunt de resultats de la funció de càlcul ha de coincidir amb el conjunt de posicions del fitxer.

La funció de càlcul es fixa en el moment de crear el fitxer calculat i s'ha de mantenir, ja que els registres s'insereixen d'acord amb el càlcul que proporciona la funció i la recerca; per tant, s'efectuarà també sota el mateix criteri.

Què cal fer, doncs, si es desitja modificar la grandària del fitxer relatiu? Això no és possible, a menys que es torni a definir la funció de càlcul per a acomodar-se a la nova grandària del fitxer i es tornin a situar els registres d'acord amb la nova funció de càlcul.

Hi ha tècniques hashing, però, que estan pensades per a poder ampliar la grandària del fitxer relatiu, acomodant els càlculs de la funció als nous requeriments, sense haver de tocar els registres existents. Estem parlant dels hashings dinàmics. Però aquestes tècniques s'escapen de l'abast d'aquest crèdit.

1.5.2. Sinònims o col·lisions

Les tècniques hashing anteriors, consistents a transformar una clau en una posició, poden donar resultats idèntics encara que els valors de partida (claus) siguin diferents. Apareixen, en aquest cas, els sinònims o les col·lisions.

> Dos o més registres són sinònims si l'aplicació de la funció de càlcul sobre cada clau dóna el mateix resultat.

Posem-ne un exemple. Considerem un fitxer calculat per a emmagatzemar detergents. Suposem que el nom del detergent n'és

identificador. Decidim la funció de càlcul següent, a partir del nom del detergent:

H(nom_detergent) = [Num(1a lletra) + Num (2a lletra) + 1] div 2

on Num (lletra) és el lloc que ocupa la lletra dins l'abecedari, començant per 1. Si considerem les lletres ç i ñ l'abecedari té 28 lletres. Suposem que el nom d'un detergent no té, en les dues primeres posicions, caràcters que no siguin lletres. Si el nom d'un detergent només té una lletra, considerarem 0 el valor de Num (2a lletra).

A partir d'aquestes consideracions i de la definició de la funció de càlcul anterior, aquesta té com a resultat valors entre 1 (un detergent de nom A) i 28 (un detergent de nom ZZ). Per tant, ens correspon considerar un fitxer relatiu de 28 registres. La figura 5 ens il·lustra l'exemple.

Figura 5. Fitxer calculat

1	
2	
3	
4	
5	
6	Ajax
7	Dixan
8	
9	Elena
10	
11	
12	
13	Ese
14	
...	
27	
28	

En aquest cas hem decidit la grandària del fitxer a partir de la funció de càlcul. En general, el procés és invers: es decideix la funció de càlcul a partir de la grandària del fitxer. (!!)

Anem introduint alguns detergents en el nostre fitxer. Per a cada detergent en calculem la posició que li pertoca dins el fitxer segons la funció de hash anterior:

Al detergent de nom Elena li pertoca la posició (6 + 13) div 2 = 9
Al detergent de nom Dixan li pertoca la posició (5 + 10) div 2 = 7
Al detergent de nom Ajax li pertoca la posició (1 + 11) div 2 = 6
Detergent de nom Ese li pertoca la posició (6 + 21) div 2 = 13
Detergent de nom Lagarto li pertoca la posició (13 + 1) div 2 = 7

S'acaba de produir una col·lisió amb el detergent de nom Dixan!

Així doncs, els detergents Dixan i Lagarto són sinònims per a la funció hash adoptada.

Què fem? Com resolem el problema?

- Solució inicial al problema dels sinònims: utilització de buckets.

Un bucket és un registre immens que conté una quantitat fixa de registres.

Terminologia anglesa

El terme bucket, que es tradueix en català com a cubell o galleda, no pot ser més exacte, ja que un bucket no és altra cosa que un cubell de registres. Ara bé, en el món de la informàtica encara no hi ha traducció per aquest terme i ens cal utilitzar el terme anglès.

La solució consisteix en implementar el fitxer relatiu com un fitxer que emmagatzemi buckets, de manera que cada bucket pugui contenir

un determinat nombre de registres. En aquest cas, en una mateixa posició del fitxer es poden emmagatzemar diferents registres (tants com capacitat tingui el bucket).

Aquesta organització no soluciona el problema, ja que tot i treballar amb buckets pot arribar el moment en què el nombre de sinònims per a una posició superi la capacitat del bucket. Per tant, no tenim altre remei que considerar l'existència dels excedents.

Els registres excedents són aquells que no poden residir a la posició que els pertoca ja que el bucket corresponent és ple per altres registres.

- Solució al problema dels excedents

– Situar-los en buckets de posicions diferents a les que els pertoquen segons la funció de càlcul. Aleshores apareixen els registres intrusos.

Un registre intrús és un registre excedent en la seva posició que està ocupant una posició destinada a altres registres.

– Situar-los en una zona especial, anomenada zona d'excedents.

La zona d'excedents és la zona destinada a contenir registres excedents. Pot estar en el mateix fitxer relatiu, després de la zona adreçada segons la funció de càlcul, o en un altre fitxer relatiu annex.

La zona principal és la zona del fitxer relatiu adreçada segons la funció de càlcul. Hi haurien d'estar, en teoria, tots els registres.

1.5.3. Gestió d'excedents

On posem els excedents? Tenim dues possibilitats: convertir-los en intrusos en la pròpia zona principal o passar-los a una zona d'excedents.

Tant si es provoquen intrusos a la zona principal com si es treballa amb una zona d'excedents, cal tenir ben definit el mètode per a trobar la posició on cal situar un excedent, ja que l'haurem d'utilitzar per a

efectuar la inserció i per a les recerques posteriors. Disposem de dues possibilitats:

– Adreçament obert, en el qual s'efectua un tractament seqüencial dins la mateixa zona principal.

– Adreçament tancat o encadenat, en el qual s'efectua un tractament amb cadenes de punters.

Adreçament obert

Aquest mètode intenta situar els excedents a la pròpia zona principal (provoca intrusos), cercant la posició mitjançant un tractament seqüencial. Com a conseqüència, les diferents operacions que cal efectuar en el fitxer tenen en compte aquesta gestió d'excedents:

Terminologia anglesa

El terme anglès per a referir-se a l'adreçament obert és open addressing. Cal tenir-lo present, ja que en molts llibres d'informàtica s'utilitza aquest nom.

1) Creació del fitxer

Tots els registres estan marcats com a verges.

2) Inserció d'un registre

Un cop calculada, mitjançant la funció de càlcul, la posició `P` on ha d'estar el registre, es mira si hi ha espai. En cas afirmatiu, s'hi col·loca. En cas negatiu, s'està davant un excedent i cal cercar-li lloc.

Els excedents s'intenten situar a la mateixa zona principal cercant una posició propera a la posició `P`, on hauria de situar-se. Sempre s'aplica la mateixa "norma" seqüencial, que pot consistir, per exemple a:

- Recórrer les posicions `P+1`, `P+2`, ..., `N`, `1`, `2`, ..., `P-1`.

- Recórrer les posicions `P+1`, `P-1`, `P+2`, `P-2`, `P+3`, `P-3`, ...

En qualsevol tipus de recorregut, aquest ha de ser circular, de manera que en arribar a la fi dels registres (per la darrera posició o per la primera posició) es continuï per l'altre extrem (per la primera posició o per la darrera posició). També cal assegurar-se que es recorren totes les posicions.

Pot succeir que el recorregut porti a la posició `P` inicial sense haver trobat espai. Això voldrà dir que el fitxer és ple. La inserció, per tant, no es pot efectuar.

3) Supressió d'un registre

Primer se n'efectua la recerca (que tractem en el punt següent) i, en trobar-lo, es marca com a esborrat. No es pot tornar a deixar verge. (!!)

Per tant, en adreçament obert tindrem registres verges, registres esborrats i registres ocupats. El procés d'inserció considerarà que en una posició hi ha espai tant si està verge com si està esborrada.

4) Recerca d'un registre

Un cop calculada, mitjançant la funció de càlcul, la posició `P` on ha d'estar el registre, es mira si hi és. En cas afirmatiu, s'ha trobat. En cas negatiu, s'està davant la recerca d'un possible excedent.

Iniciarem el mateix recorregut seqüencial utilitzat en efectuar les insercions, el qual finalitzarà per un dels motius següents:

- Es troba el registre cercat.
- Es torna a la posició de partida o es troba un registre verge. En aquestes situacions, el registre cercat no és a el fitxer.

Cal observar que el fet de trobar un registre verge implica la fi de la recerca amb la conclusió que no hi ha un tal registre. Per aquest motiu, és fonamental el fet de senyalar els registres esborrats amb una marca diferent dels registres verges.

La figura 6 ens exemplifica la situació. Hi podem observar on s'ha situat el registre R3. La funció de càlcul donava la posició 3, la qual era plena. Segons el tractament seqüencial amb la norma `P+1`, `P+2`, ... s'ha trobat lloc a la posició 5 i s'hi col·loca. És un intrús.

Figura 6. Fitxer calculat amb adreçament obert.

1		V
2		V
3	R1 (hash = 3)	O
4	R2 (hash = 4)	O
5	R3 (hash = 3)	O
6		V
7	R4 (hash = 7)	O
8		

Suposem que es vol cercar un registre R (no existent al fitxer) que segons la funció de càlcul hauria d'estar a la posició 3. S'accedeix a la posició 3 i es comprova que, tot i estar ocupada, no conté el registre cercat. S'inicia el procés de recerca (tractament seqüencial amb la norma `P+1`, `P+2`, ...) passant per les posicions 4 (ocupada per un registre diferent del cercat), 5 (ocupada per un registre diferent del cercat) i 6 (registre verge!). Conclusió: no hi ha el registre cercat.

Suposem que se suprimís el registre R2 (posició 4) i es tornés a deixar la posició com a verge. En cercar el registre R3, la recerca començaria en la posició 3 i finalitzaria en la posició 4 per trobar un registre verge, amb la conclusió errònia que no hi havia el registre cercat. Aquí es pot observar la necessitat de marcar els registres esborrats de manera diferent dels registres verges. Així doncs, la supressió del registre R2 ha d'implicar deixar la seva posició com a esborrada i, en aquest cas, la recerca del

registre R3 passaria per les posicions 4 (esborrada) i 5 (ocupada pel registre cercat).

El mètode d'adreçament obert té dos inconvenients:

- Pot quedar ple si la previsió de registres és inferior al nombre real. En aquest cas caldrà tornar a definir la funció de càlcul per a una previsió superior i regenerar el fitxer calculat amb la nova definició.

- Pot degenerar en cas que es produeixin moltes supressions. En aquest cas caldrà regenerar el fitxer eliminant les posicions marcades com a esborrades.

D'altra banda, hi ha dues millores importants que es poden aplicar a aquest mètode en la creació del fitxer si es coneix el conjunt inicial de registres a inserir. (!!)

En efecte, sovint es defineix un fitxer calculat i, de bon principi, s'ha d'efectuar la càrrega d'un conjunt inicial de registres. En aquesta situació es poden efectuar les insercions inicials sota les consideracions següents, que representen una millora respecte al mètode genèric d'inserció de l'adreçament obert.

- Càrrega en dues passades. Aquest mètode consisteix a inserir, en una primera passada pel conjunt de registres a carregar, els registres que troben espai a la posició que els pertoca segons la funció de càlcul. En una segona passada es col·loquen els registres excedents, els quals seran intrusos.

  Deixant els registres excedents per a la segona passada es minimitza la quantitat d'intrusos. Si no es fa així, un intrús pot provocar que un registre que hauria d'estar en la posició ocupada per l'intrús es converteixi en intrús en una altra posició.

- Càrrega en dues passades per freqüència d'utilització. Aquest mètode s'assembla molt a l'anterior, però amb l'excepció que es té en compte la freqüència d'ús dels registres a inserir, de manera que abans de la càrrega amb dues passades s'efectua una ordenació del conjunt de registres a carregar segons la seva freqüència d'utilització. D'aquesta manera, s'intenta aconseguir que els registres més emprats ocupin la posició que els pertoca per tal de minimitzar el temps de recerca.

  No cal dir que per a poder aplicar aquest mètode es necessita informació suplementària referent al grau d'utilització de cada registre.

Hi ha, també, dues millores importants que es poden aplicar al mètode d'adreçament obert durant la gestió del fitxer. (!!)

- Treballar amb intrusos errants, és a dir, tota inserció de registre que es trobi la seva posició ocupada per un intrús l'ha de fer fora, ha d'ocupar la seva posició i ha d'obligar a inserir de nou el registre intrús.

  Aquesta gestió provoca que les insercions siguin lentes, però en canvi produeix una substancial millora en les consultes.

  Cal tenir una precaució important: abans de fer fora l'intrús per a efectuar la inserció del registre cal tenir la seguretat que hi ha espai per a recol·locar l'intrús.

- Efectuar les supressions tornant a deixar el fitxer com a verge i provocant immediatament un recàlcul sobre tots els registres existents a continuació, segons la norma seqüencial establerta, fins al proper registre verge.

  Aquesta gestió provoca que les supressions siguin lentes, però garanteix que hi hagi els intrusos imprescindibles; altrament, si no s'apliqués aquesta millora, un registre podria ser un intrús perquè en el moment en què va ser inserit hi havia un sinònim que ocupava la seva adreça mentre que en l'actualitat aquell sinònim ha estat esborrat i el registre no ha estat recol·locat.

Adreçament tancat o encadenat

En aquest mètode, els excedents estan encadenats i, per tant, cal mantenir les cadenes en les diverses operacions d'actualització del fitxer. Una recerca finalitzarà en trobar el valor cercat o el final de la cadena.

Terminologia anglesa

El terme anglès per a referir-se a l'adreçament tancat o encadenat és closed addressing. Cal tenir-lo present, ja que en molts llibres d'informàtica s'utilitza aquest nom.

Hi ha diferents tipus d'adreçament tancat que podem classificar segons diferents criteris. La taula 4 ens en mostra un resum.

1) Segons que les cadenes siguin a nivell de registre o a nivell de bucket.

a) Cadenes a nivell de registre. Cada registre conté un punter al bucket i al registre col·locat dins el bucket on hi ha d'haver el sinònim següent.

- Amb zona d'excedents. En inserció, el sinònim excedent es col·loca en el primer registre lliure de la zona d'excedents. En un bucket de la

zona d'excedents hi pot haver registres no sinònims entre si, però els encadenaments es donen a nivell de registre, de manera que cada registre apunta a un altre registre.

Taula 4. Classificació dels mètodes d'adreçament tancat per la gestió d'excedents en fitxers calculats

Segons les cadenes siguin a nivell de registre o a nivell de bucket			**Segons que els registres recorreguts seguint les cadenes siguin o no, tots ells, sinònims entre sí**
Cadenes a nivell de registre	Disposant de zona d'excedents		Mètodes separate list
	Sense zona d'excedents (només zona principal)		
Cadenes a nivell de bucket	Disposant de zona d'excedents	Utilitzant buckets especialitzats	
		Utilitzant buckets no especialitzats	Mètodes coadlesced list
	Sense zona d'excedents (només zona principal)		

- Només amb zona principal. En inserció, es cerca un registre lliure a la zona principal, on es col·loca el sinònim excedent. Per tant, un bucket de la zona principal pot contenir, d'una banda, registres col·locats en el bucket que els pertoca i, de l'altra, registres intrusos.

b) Cadenes a nivell de bucket. Cada bucket conté un punter al bucket on hi ha d'haver el sinònim següent.

- Amb zona d'excedents.

– Utilitzant buckets especialitzats, és a dir, buckets que continguin registres tots ells sinònims entre si.

– Utilitzant buckets no especialitzats, és a dir, buckets que poden contenir registres no sinònims entre si.

- Només amb zona principal. En aquest cas, els buckets de la zona principal forçosament no poden ser especialitzats.

2) Segons que els registres recorreguts seguint els encadenaments, siguin o no, tots ells, sinònims entre si.

a) Separate list. Tots els registres a què s'accedeix seguint l'encadenament són sinònims entre si.

b) Coalesced list. S'accedeix a registres no sinònims entre si seguint l'encadenament.

2. Gestió de fitxers seqüencials indexats

Hem arribat, per fi, al moment culminant en l'estudi dels sistemes gestors de fitxers: l'accés seqüencial indexat.

> Un fitxer seqüencial indexat és aquell que permet l'accés seqüencial i directe per valor mitjançant diverses vies d'accés.

Els fitxers seqüencials indexats s'implementen diferenciant les dades (registres que emmagatzemen) dels índexs (valors claus que permeten l'accés directe i seqüencial per valor) i segons siguin els índexs emprats, podem establir classificacions dels fitxers seqüencials indexats.

De forma similar a l'aprenentatge de la resta de SGF ens convé conèixer les operacions teòriques que proporcionen els SGF seqüencials indexats i, posteriorment, els algorismes bàsics de gestió.

El llenguatge C no incorpora cap SGF seqüencial indexat i, clar, podríem implementar-lo, però no cal dedicar-hi ni un minut de temps donat que en el mercat hi ha SGF seqüencials indexats que permeten ser utilitzats des de diferents llenguatges com C, Pascal, Fortran, etcètera. Per tant, convé saber com des del llenguatge C podem gestionar algun d'aquests SGF seqüencials indexats.

Un darrer comentari important abans d'iniciar l'estudi detallat dels SGF seqüencials indexats: en l'actualitat, poc, per no dir gens, s'utilitzen els SGF de cap tipus (ni els seqüencials, ni els relatius, ni els seqüencials indexats) en les aplicacions informàtiques utilitzades a les empreses. Per tant, per què dedicar-hi temps? Per una raó molt clara: les actuals aplicacions informàtiques gestionen sistemes gestors de bases de dades (SGBD) que han sorgit com evolució dels SGF seqüencials indexats i ens convé conèixer a fons el funcionament dels SGF per a treure'n un millor profit dels SGBD. (!!)

2.1. Distinció entre dades i índexs

Per a aconseguir l'accés directe i seqüencial per valor a un fitxer de dades és necessària la utilització d'un índex consistent en un conjunt de notes, una per a cada registre del fitxer, que conté:

- El valor del camp pel qual es desitja efectuar l'accés directe.

- Un punter, registre de direcció, que permet l'accés immediat al registre del fitxer.

L'índex (conjunt de notes) cal conservar-lo sempre ordenat pel camp que indexa, de manera que la recerca directa per valor pugui ser ràpida i es pugui efectuar el recorregut seqüencial pel valor del camp. La funció de l'índex és anàloga a la de l'índex d'un llibre: facilitar la recerca d'una determinada paraula o d'un tema.
El tipus més elemental d'índex és el lineal simple. És el tipus d'índex propi d'una biblioteca tradicional on podem cercar els llibres per diferents vies d'accés:

- Classificació per autors consistent en un arxiu de fitxes (una per llibre) ordenades a partir del nom de l'autor. Cada fitxa inclou la ubicació del llibre.

- Classificació per títols consistent en un arxiu de fitxes (una per llibre) ordenades a partir del títol. Cada fitxa inclou la ubicació del llibre.

- Classificació per temes consistent en un arxiu de fitxes (una per llibre) ordenades a partir de la temàtica. Hi pot haver diferents nivells: tema, subtema, categoria... Cada fitxa inclou la ubicació del llibre.

L'exemple presentat fa referència a una situació real a les biblioteques no informatitzades. Imaginem un fitxer (en sistema informàtic) que conté dades de persones com el de la figura 7.

Figura 7. Exemple de fitxer informàtic que emmagatzema dades de persones.

	Número	Cognom	Nom	Data naixement	Pes	Altura	Sexe
1	7893	Ollé	Teresa	04.12.1985	45	150	D
2	4536	Esteve	Guifré	12.04.1980	60	170	H
3	2375	Miralles	Oriol	24.05.1979	62	173	H
4	9823	Brito	Júlia	23.09.1983	50	165	D
5	1287	Pelfort	Anna	15.07.1987	53	160	D
6	7834	Coromina	Jaume	10.10.1980	70	175	H

Podem perfectament suposar que el fitxer de la figura 7 facilita l'accés directe per posició. Observem que les posicions estan numerades de l'1 al 6.

Ens interessa poder tenir accés seqüencial i directe pel camp número i pel camp cognom. Hem de considerar dos índexs diferents, com mostra la figura 8.

Figura 8. Índexs per "Número" i per "Cognom" per al fitxer de la figura 7.

	Índex per "Número"	
	Número	Punter
1	1287	5
2	2375	3
3	4536	2
4	7834	6
5	7893	1
6	9823	4

	Índex per "Cognom"	
	Cognom	Punter
1	Brito	4
2	Coromina	6
3	Esteve	2
4	Miralles	3
5	Ollé	1
6	Pelfort	5

Observem que amb aquesta organització podem aconseguir:

- Accés directe per valor, ja que donat un valor clau (número o cognom), el cerquem de manera ràpida en l'índex corresponent. Si els índexs són, per exemple, taules, en estar ordenades, podem aplicar un algorisme de recerca per dicotomia, que és d'ordre logarítmic.

- Accés seqüencial per valor, ja que si es vol efectuar un recorregut seqüencial de les persones classificades per un dels camps número o cognom, simplement s'ha de fer un recorregut seqüencial per l'índex corresponent, on ens trobem els números o cognoms classificats convenientment i amb un punter que ens adreça al fitxer on hi ha les dades.

- Accés dinàmic, que permet accedir directament per valor a un registre i, a partir d'aquest, realitzar un accés seqüencial fins al final del fitxer o fins que es deixi de verificar la condició de recerca.

Fixem-nos en que aquesta organització coincideix amb les taules de consulta com a mètode de transformació clau-adreça en els fitxers calculats, mètode que té múltiples inconvenients (problemes de memòria si el fitxer té un gran nombre de registres i problemes d'eficiència per al manteniment dels índexs). En la realitat, els índexs no seran d'aquest estil, però aquestes "taules de consulta" van molt bé per a entendre què es pretén amb la indexació.

Suposo que veiem clares les facilitats de treball que hi ha amb aquest tipus d'organització. Ara bé, els fitxers seqüencials i relatius emmagatzemaven dades i, ocasionalment, marques de l'estat del registre (verge, ocupat, esborrat...). Però, ara, els fitxers han d'emmagatzemar les dades. On fiquem els índexs (vies d'accés) corresponents?

Els índexs també s'emmagatzemen en fitxers. Hi ha diferents possibilitats i en el mercat us podeu trobar SGF amb les diverses variants:

a) Un fitxer per a les dades i un fitxer per a cada índex. Parlem llavors de fitxer de dades i de fitxers índexs.

Una organització d'aquest estil és la que seguia l'obsolet SGF comercial dBase III, on les dades apareixen en un fitxer que físicament tenia extensió dbf i cada índex estava en un fitxer d'extensió ndx.

Una altra organització d'aquest estil la trobem en l'SGF comercial que incorporava el llenguatge de programació Clipper, on les dades apareixien en un fitxer d'extensió dbf (idèntic format de dBase III) i cada índex estava en un fitxer d'extensió ntx (no compatible amb ndx).

b) Un fitxer per a les dades i un fitxer per a tots els índexs. Parlem llavors de fitxer de dades i de fitxer multiíndex.

L'SGF dBase IV facilitava una organització d'aquest estil: el fitxer de dades tenia extensió dbf (però és format dBase IV) i el fitxer multiíndex tenia extensió mdx. Aquesta organització permetia tenir diversos fitxers mdx i, per compatibilitat amb el seu predecessor, també permetia els índexs ndx. En un fitxer mdx hi podia haver fins a 50 vies d'accés diferents.

c) Un únic fitxer que conté les dades i els índexs.

L'SGF Btrieve utilitzava aquesta organització, la qual té un avantatge important: tota la informació necessària per a la gestió del fitxer indexat resideix en un únic fitxer. Ara bé, això també es pot considerar un inconvenient ja que les dades i els índexs no estan separats físicament, de manera que si interessa agafar únicament les dades caldrà fer un procés d'exportació del fitxer indexat cap a un fitxer seqüencial o relatiu.

Per a finalitzar aquest apartat dedicat a la distinció entre les dades i els índexs, convé deixar clars un seguit de punts:

- Qualsevol camp del fitxer de dades pot servir de base per a construir un índex.

- Una combinació de camps del fitxer pot servir de base per a construir un índex. Per exemple, per a obtenir un llistat ordenat alfabèticament, l'índex ha de ser constituït pels camps cognoms i nom.

- Qualsevol índex pot utilitzar-se tant per l'accés directe, en funció del camp indexat, com per l'accés seqüencial, segons la seqüència d'aquest camp.

- Un determinat fitxer pot tenir associats diversos índexs, per un camp o per una combinació de camps.

- Quan es modifiquen les dades és necessari actualitzar tots els índexs associats a les dades. És una tasca feixuga, ja que en tot moment hi ha d'haver una correspondència entre les dades i els seus índexs.

2.2. Classificacions d'índexs

En principi podem classificar els índexs segons dos criteris:

1) En funció de la quantitat d'entrades relatives a les dades que emmagatzemen, tenim índexs densos i índexs escassos.

a) Índex dens o exhaustiu: hi ha un registre índex (entrada) per a cada valor del camp clau en les dades. És el tipus d'índex lineal similar a les taules de consulta com a mètode de transformació clau-adreça en els fitxers calculats i que s'acostuma a mostrar com a exemple d'indexació.

b) Índex escàs o selectiu: es creen registres índexs per a alguns dels registres del camp clau en les dades. És el tipus d'índex utilitzat en els veritables SGF seqüencials indexats.

2) En funció de la quantitat de nivells que componen internament l'índex, tenim índexs simples i índexs de nivells múltiples.

a) Índex simple: totes les entrades a l'índex estan en un únic nivell. És el tipus d'índex lineal similar a les taules de consulta com a mètode de transformació clau-adreça en els fitxers calculats i que s'acostuma a mostrar com a exemple d'indexació.

b) Índex multinivell: les entrades en l'índex s'agrupen en blocs i es pot crear un índex de segon nivell per a proporcionar trajectòries més curtes al primer nivell, encara que s'utilitzin índexs escassos a causa de la gran quantitat de registres de dades. Poden crear-se índexs de tres o més nivells si el segon nivell es fa molt gran i no pot ser analitzat seqüencialment. És el tipus d'índex utilitzat en els SGF seqüencials indexats.

De fet, els SGF més moderns utilitzen tècniques d'índexs escassos i multinivell. Podem precisar encara més i dir que els SGF moderns utilitzen organitzacions basades en arbres B i en les seves variants.

Arbres B

Els arbres (i en particular els arbres B) són un tipus de dada dinàmica (com també ho són les llistes, les piles i les cues) que s'estudien en textos especialitzats en "Estructures de dades dinàmiques".

2.3. Operacions teòriques

La majoria d'SGF seqüencials indexats aporten les operacions semblants a les que presentem a continuació en pseudocodi.

```
funció obrir_fi (f: fitxer_ind de T, F: cadena, mode: caràcter) retorna enter;
/* mode: 'C' (consulta), 'A' (actualització) */

funció tancar_fi (f: fitxer_ind de T) retorna enter;
/* realitza les gravacions pendents i tanca el fitxer */

funció afegir_fi (f: fitxer_ind de T, r: T) retorna enter;
/* intenta afegir el registre r a el fitxer */

funció modificar_fi (f: fitxer_ind de T, r: T) retorna enter;
/* modifica el darrer registre llegit */

funció esborrar_fi (f: fitxer_ind de T) retorna enter;
/* esborra el darrer registre llegit */

funció llegir=_fi (f: fitxer_ind de T, var r: T, via: cadena, clau: TI) retorna enter;
/* cerca el primer registre per l'índex via amb valor clau i, si el troba, omple r */

funció llegir>=_fi (f:fitxer_ind de T, var r:T, via:cadena, var clau:TI) retorna enter;
/* cerca el primer registre per l'índex via amb valor clau o l'immediat superior, si no existeix un
valor igual i, si el troba, deixa el valor a r i el corresponent valor de la clau a clau */

funció llegir>_fi (f:fitxer_ind de T, var r:T, via:cadena, var clau:TI) retorna enter;
/* cerca el primer registre per l'índex via amb el valor immediat superior a clau i, si el troba,
deixa el valor a r i el corresponent valor de la clau a clau */

funció llegir<=_fi (f:fitxer_ind de T, var r:T, via:cadena, var clau: TI) retorna enter;
/* cerca el darrer registre per l'índex via amb valor clau o l'immediat anterior, si no existeix un
valor igual i, si el troba, deixa el valor a r i el corresponent valor de la clau a clau */

funció llegir<_fi (f:fitxer_ind de T, var r:T, via:cadena, var clau:TI) retorna enter;
/* cerca el darrer registre per l'índex via amb el valor immediat anterior a clau i, si el troba,
deixa el valor a r i el corresponent valor de la clau a clau */

funció llegirinf_fi (f:fitxer_ind de T, var r:T, via:cadena, var clau:TI) retorna enter;
/* cerca el primer registre per l'índex via i, si el troba, deixa el valor a r i el corresponent
valor de la clau a clau */

funció llegirsup_fi (f:fitxer_ind de T, var r:T, via:cadena, var clau:TI) retorna enter;
/* cerca el darrer registre per l'índex via i, si el troba, deixa el valor a r i el corresponent
valor de la clau a clau */

funció llegirseg_fi (f:fitxer_ind de T, var r:T, via:cadena, var clau:TI) retorna enter;
/* cerca el següent registre per l'índex via i, si el troba, deixa el valor a r i el corresponent
valor de la clau a clau; es basa en la darrera lectura efectuada i cal tenir el camp clau ple amb la
part corresponent al darrer registre llegit */

funció llegirant_fi (f:fitxer_ind de T, var r:T, via:cadena, var clau:TI) retorna enter;
/* cerca l'anterior registre per l'índex via i, si el troba, deixa el valor a r
i el corresponent valor de la clau a clau; es basa en la darrera lectura efectuada i cal tenir el
camp clau ple amb la part corresponent al darrer registre llegit */
```

Observem que:

- Totes les instruccions de lectura tenen, a més de l'argument corresponent al fitxer intern i de l'argument corresponent a la variable registre a omplir, els arguments següents:

– `clau`, que és de tipus `TI` (sigla de tipus índex), tipus variable en funció de l'índex pel qual s'estigui accedint, ja que els diferents índexs que existeixen en un fitxer seqüencial indexat poden ser, evidentment, de diferent mena. Aquest argument ha d'omplir-se amb el valor que correspongui en totes les operacions de lectura, exceptuant `llegirinf_fi` i `llegirsup_fi`, ja que en aquestes l'SGF no necessita cap valor. D'altra banda, observeu que l'argument és passat per referència, ja que l'SGF l'omple amb el valor clau del registre llegit a excepció de l'operació `llegir=_fi`, en la qual l'SGF no l'omple perquè el valor passat en la crida no pot diferir del valor clau trobat ja que es tracta de l'operació `llegir=_fi`.

– `via`, que és de tipus `cadena`, és la variable utilitzada per a referir-se a l'índex que cal fer servir entre tots els índexs que pot tenir el fitxer indexat. En pseudocodi podem suposar que si l'índex a utilitzar és format pels camps de noms `camp_A`, `camp_B` i `camp_C`, utilitzaríem la cadena constituïda per `camp_A+camp_B+camp_C` per a indicar que cal fer servir aquest índex com a via d'accés.

Com podem indicar la via d'accés?

Els SGF utilitzen diferents mecanismes per a indicar la via d'accés.

N'hi ha que numeren els índexs, i en les diferents operacions el paràmetre via no és altra cosa que un valor natural que indica el número de via d'accés. *Btrieve* utilitza aquest mecanisme.

Altres SGF obliguen a batejar els índexs en el moment de la seva creació. És el cas de les diverses variants de *dBase*.

- Les operacions de lectura `llegirant_fi`, `llegir<_fi` i `llegir<=_fi` recorren l'índex en direcció descendent, és a dir, des de la darrera entrada llegida cap enrere segons l'ordre de l'índex. No s'ha de confondre amb l'ordre de classificació que constitueix l'índex. Així, per exemple, si tenim un índex format per nombres naturals classificats descendentment de la forma 50, 43, 22, 10, 5 i efectuem una operació `llegirant_fi` després d'haver efectuat una operació `llegir=_fi` del valor 22, l'SGF ens lliurarà el registre de valor clau 43, ja que és el que hi ha abans del valor 22 dins l'índex i, evidentment, el valor 43 no és menor que el valor 22.

- Hi ha índexs que admeten valors duplicats. En aquest cas, fixeu-vos que:

– Les operacions `llegirsup_fi`, `llegir<_fi` i `llegir<=_fi` ens retornen sempre el darrer duplicat existent.

– Les operacions `llegirinf_fi`, `llegir>_fi` i `llegir>=_fi` ens retornen sempre el primer duplicat existent.

– Les operacions `llegirant_fi` i `llegirseg_fi` recorren els diversos duplicats que hi pugui haver per sota i per sobre del valor actual.

• Molts SGF acostumen a tenir per a cada operació `llegir...` l'homònima `ncllegir...` (`només_clau_llegir...`), que consisteix a cercar únicament el valor clau corresponent, de manera que s'optimitza l'execució ja que no s'ha d'accedir als blocs de disc on hi ha les dades.

Aquestes són, doncs, les operacions i observacions principals a tenir en compte en la gestió d'SGF seqüencials indexats. Però per a familiaritzar-se amb els SGF seqüencials indexats el millor és conèixer a fons com funcionen realment i per aconseguir-ho ens endinsarem en el món de l'SGF Btrieve.

2.4. SGF Btrieve

Btrieve és un sistema gestor de fitxers que permet una gestió eficaç de fitxers amb accés seqüencial i directe per posició i per valor. (!!)

Per tant, és un SGF que engloba els SGF relatius (accés seqüencial i directe per posició) i els SGF seqüencials indexats (accés seqüencial i directe per valor).

No és un manual

El material de Btrieve que presentem pot produir la sensació de *manual de l'SGF Btrieve*. No és exactament així. Hi ha molts conceptes relatius a aquest SGF i els que presentarem s'han escollit en base a:

- Ser conceptes genèrics que us podeu trobar, de forma similar, en qualsevol SGF seqüencial indexat
- Ser conceptes necessaris per a poder desenvolupar programes en llenguatge C efectuant crides a l'SGF Btrieve.
- Ser conceptes molt vinculats al funcionament dels actuals SGBD, fet que indica que Btrieve va arribar a ser un SGF molt avançat.

Per últim, tenir en compte que la informació presentada es basa en la versió 5.10a de l'any 1990 per a plataforma DOS. És una versió obsoleta que no té cap interfície gràfica per a definir ni administrar els fitxers, però aporta tot el necessari pels continguts de conceptes i de procediments a treballar.

2.4.1. Una mica d'història

Btrieve ha estat propietat de tres companyies diferents: SoftCraft, Novell i Btrieve Technologies, Inc. i totes elles han contribuït al seu desenvolupament.
El producte va néixer el febrer del 1982 de les mans de SoftCraft. En febrer del 83 va començar la saga de les versions 2.x i quan la versió 2.0 del sistema operatiu MS-DOS va proporcionar suport per fitxers, va aparèixer la versió 3.0 de Btrieve. Un més després de l'aparició de la versió 3.1 de MS-DOS en març del 85, apareix Btrieve 3.1 C/S que suportava accessos en xarxa i permetia el treball client/servidor. La versió 4.0 de Btrieve va aparèixer el febrer del 86 i la posterior versió 4.1 ja donava suport als principals tipus de dades i permetia índexs suplementaris. Tot i que Btrieve era bastant popular, no es diferenciava molt del SGF dBase (mal anomenat SGBD) que havia obtingut molta popularitat.

En 1987 Novel va comprar Softcraft i continua l'evolució de Btrieve. A principis del 1988 apareix Btrieve 5.0 amb més millores. Es van succeint diverses versions fins la 5.1 que apareix en 1990 i per a diferents plataformes: MS-DOS, OS/2 i MS-Windows. En 1992 Novell presenta la versió 6.0 de Btrieve la qual és incompatible amb les versions anteriors fet que provoca un desencís en la comunitat informàtica que utilitza Btrieve.

Passa un cert temps sense que Btrieve sigui correctament promocionat per Novell i aquesta decideix, en 1994, transferir la titularitat de Btrieve a la nova empresa Btrieve Technologies, Inc. dirigida per les mateixes persones que havien dirigit Softcraft. Btrieve va ser totalment reescrit i en juliol del 94 apareix la versió Btrieve 6.15 per a MS-DOS, MS-Windows i OS/2. En 1995 la versió 6.15 ja es pot utilitzar en Novell Netware, Windows NT Server i Windows NT/95.

En 1996, l'empresa canvia de nom i passa a anomenar-se Pervasive Software. La nova empresa allibera el seu nou producte Pervasive.SQL v7 que inclou Btrieve 7.0 i noves prestacions per a les principals plataformes existents. Posteriorment han aparegut noves versions: Pervasive.SQL 2000/2000i, Pervasive.SQL v8 (l'any 2002), Pervasive PSQL v9 (l'any 2005) i a finals de l'any 2007 ha anunciat la propera aparició de la versió 10.

2.4.2. Característiques de Btrieve

Les característiques de gestió que aporta Btrieve i que permeten comparar-lo amb altres SGF comercials són: ‼

1) Permet accés per claus múltiples

Btrieve permet accedir a un fitxer per vies o claus diferents. L'SGF manté un índex diferent per a cada via i emmagatzema els registres en ordre ascendent, en funció dels valors de la clau indicada.

Aquesta característica l'acostumen a incorporar la majoria d'SGF indexats.

2) Efectua el manteniment automàtic de claus

Btrieve actualitza automàticament els índexs que apunten als registres d'un fitxer quan hi ha operacions d'afegir nous registres, esborrar registres i/o modificar registres.

És evident que un SGF ha d'incorporar aquesta característica. ¿Us imagineu que el programador hagués de programar els canvis en els diversos índexs del fitxer davant d'altes, baixes i/o modificacions? En aquest cas, l'SGF només ens serviria per a fer recerques.

3) Gestiona claus duplicades, modificables, segmentades i nul·les

La duplicitat permet que un únic valor de clau identifiqui un subconjunt de registres dins d'un fitxer. Quan no es permeten claus duplicades, qualsevol intent d'inserir un registre amb una clau ja existent és interceptat per Btrieve.

La modificabilitat permet la variació del valor d'una clau dins d'un registre. Si no es permet fer modificacions, Btrieve intercepta qualsevol intent de canvi del valor d'una clau.

La segmentació permet que el valor d'una clau sigui format per caràcters no consecutius dins el registre. Cada conjunt consecutiu de caràcters s'anomena segment de la clau.

Una clau pot contenir un valor nul. Un valor nul és el contingut d'un byte, definible per l'usuari, de manera que quan es troba en totes les posicions del camp de clau, en afegir un registre, no s'actualitza el corresponent índex de clau. En accés seqüencial per valor del camp de clau en què s'ha definit un valor nul, se salten tots els registres que tenen aquest valor nul.

En les claus segmentades, els diversos segments podran ser de diferents tipus de dades, però tots han de tenir el mateix criteri de duplicitat, modificabilitat i nul·litat. En cas que tots permetin l'existència d'un valor nul, aquest pot ser diferent en els diversos segments.

4) Permet fitxers repartits

Permet que un fitxer es reparteixi en dos discos. Quan un fitxer s'ha repartit, les seves parts han de residir sempre en els discos respectius, és a dir, no es pot desfer el repartiment.

5) Treballa amb memòria intermèdia d'entrada/sortida

Nomenclatura específica

Btrieve utilitza una nomenclatura específica per a referir-se a alguns conceptes genèrics que ja coneixem amb un altre nom. Així, en lloc de *buffer* utilitza àrea de memòria intermèdia i en lloc de bloc utilitza pàgina.

La memòria intermèdia d'entrada/sortida és un àrea de memòria interna reservada per a emmagatzemar les pàgines llegides des del disc. La seva grandària es defineix amb un paràmetre en carregar Btrieve. Com més grandària, major rendiment.

En rebre una petició de lectura d'un registre, Btrieve comprova si la pàgina que el conté ja és a la memòria intermèdia. En aquest cas, es traspassa el registre al programa d'aplicació. En cas contrari, es llegeix la pàgina del disc a la memòria intermèdia, abans que el registre llegit es passi al programa. Si no hi ha espai disponible a la memòria intermèdia per a llegir una pàgina, un algorisme UMR (ús menys recent) determina quina pàgina ha de ser substituïda.

En condicions normals d'operació, quan es rep una petició d'escriure un registre al disc, la pàgina de memòria intermèdia que conté el registre és modificada i immediatament escrita al disc. La pàgina modificada continua a la memòria intermèdia fins que l'algorisme UMR determini que ha de ser canviada per una altra pàgina. Ara bé, és possible obrir un fitxer en mode accelerat, la qual cosa fa que les escriptures no siguin necessàriament transferides al disc immediatament.

6) Incorpora controls d'integritat

Vegem que disposa de controls d'integritat de les dades a diferents nivells.

a) A nivell d'un fitxer. Btrieve protegeix contra situacions d'inconsistència que puguin originar-se en un error del sistema (per exemple, un tall de llum) mentre s'estan fent canvis en el fitxer. Es manté la integritat eliminant qualsevol canvi incomplet que s'hagi aplicat. Aquest procediment de recuperació és automàtic en la següent obertura del fitxer. Això no és possible si la part que s'espatlla és el disc.

b) A nivell de la base de dades (fitxers relacionats). Btrieve permet definir un conjunt d'operacions lògicament relacionades, englobant fins a 12 fitxers diferents, com una única transacció mitjançant dues instruccions, una d'"inici de transacció" i una altra de "final de

transacció". Btrieve assegura la integritat de la transacció consolidant les operacions al final. En cas que entre un "inici" i un "final" el sistema tingui una fallada, Btrieve elimina automàticament les operacions portades a terme en la transacció incompleta la pròxima vegada que s'accedeix als fitxers afectats. A més, es pot forçar a eliminar una transacció incompleta amb una instrucció d'"avortar transacció" abans que la transacció finalitzi.

7) Permet fitxers de grandària il·limitada

La grandària màxima d'un fitxer que pot gestionar Btrieve 5.10a és d'aproximadament 4.000 Mb. Per tant, la grandària dels fitxers estava marcada més pel suport físic i pel sistema operatiu que no pas per la limitació de gestió del mateix SGF (pensem en les grandàries dels suports físics a l'any 1990!)

No hi ha cap limitació referent als valors diferents de clau i a la quantitat de registres en un fitxer.

8) Facilita utilitats per creació i manteniment de fitxers des del sistema operatiu

9) Gestiona l'accés en entorns multiusuari i multitasca

Btrieve permet accessos concurrents al mateix fitxer en aplicacions executables en entorns multiusuari o de xarxa local protegint l'estructura física del fitxer dels possibles danys que pogués causar una actualització simultània de registres. A més, disposa de procediments de gestió de la concurrència, incloent-hi el bloqueig de registres.

Podrem comprovar aquests conceptes si disposem de Btrieve per a MS-DOS? I tant! Només cal obrir diferents sessions DOS, carregar l'SGF en cada sessió i executar els programes simultàniament. Podrem comprovar el funcionament del control de transaccions i dels bloqueigs de registres.

10) Disposa de restriccions d'accés a fitxers

Btrieve permet també protegir un fitxer contra accessos no autoritzats, mitjançant l'assignació d'una paraula de pas per aquell fitxer. La paraula de pas pot servir per a qualsevol tipus d'accés o només per a prevenir l'escriptura en el fitxer. La paraula de pas també pot utilitzar-se per a encriptar les dades del fitxer.

11) Permet operatòria des de diferents llenguatges de programació

Btrieve pot utilitzar-se en aplicacions escrites en diferents llenguatges, i fins i tot es pot escriure una rutina en llenguatge d'assemblador per a utilitzar Btrieve des de qualsevol llenguatge no suportat directament.

La manipulació de registres en un fitxer Btrieve es realitza amb crides des de les aplicacions a una única funció, el codi de la qual ha de ser accessible des de l'aplicació.

2.4.3. Fitxers Btrieve: registres, claus, estructura física.

Els fitxers gestionats pel SGF Btrieve tenen una determinada estructura i contingut i s'acostuma a parlar de fitxers Btrieve per a referir-nos als fitxers gestionats per l'SGF Btrieve.

No intenteu accedir als fitxers Btrieve directament des del sistema operatiu. Si ho feu en mode de lectura no veureu la informació correcta. Si ho feu en mode d'edició, podeu provocar destrosses que impossibilitin la gestió posterior del fitxer amb l'SGF Btrieve.

Per tal de conèixer els fitxers Btrieve, en primer lloc presentarem els seus registres, posteriorment tractarem les seves claus i, finalment, veurem l'estructura física del fitxer.

Registres

Els registres poden tenir una longitud fixa o ser formats per una part fixa i una part variable. Totes les claus que es defineixin en el fitxer han de ser a la part fixa, la qual pot arribar a ser de 4.090 bytes. Els registres de longitud variable poden arribar a mesurar 64 KB.

Claus

Les claus en Btrieve no han de coincidir necessàriament amb un o més camps del registre. Qualsevol conjunt de caràcters del registre pot ser una clau. Diferents claus poden aixafar-se dins d'un mateix registre.

La longitud total màxima d'una clau és de 255 caràcters. En el cas de claus segmentades, la longitud total de la clau equival a la suma de les longituds dels seus segments.

Btrieve permet definir fins a un màxim de 24 segments de clau per un fitxer. Un fitxer pot contenir, per tant, una clau amb 24 segments, 24 claus amb un segment cadascuna o qualsevol combinació intermèdia. El fet que un fitxer pugui tenir 24 segments de clau depèn de certes característiques del fitxer, que veurem més endavant.

En definir un segment de clau s'ha d'identificar amb un dels tipus de dades que Btrieve aporta perquè pugui ser indexat en l'ordre adequat.

Penseu que els tipus de dades que Btrieve gestiona són pròpiament de Btrieve i no han de coincidir necessàriament amb els tipus de dades que gestiona el llenguatge en què codifiquem programes que accedeixin a fitxers Btrieve. (!!)

Els tipus de dades aportats per Btrieve són:

a) De tipus cadena

- string

  Correspon al tipus de cadena que interpreta cada octet com un caràcter i que ordena d'acord amb l'ordre lexicogràfic.

- lstring

  Correspon al tipus de cadena que té el mateix sistema de gestió que les cadenes del llenguatge de programació Pascal, és a dir, una tira de caràcters, amb l'excepció que el primer byte de la tira correspon a la representació binària de la longitud x de la cadena. Btrieve la utilitza per a considerar com a cadena el conjunt de x caràcters des del segon byte.

- zstring

  Correspon al tipus de cadena que té el mateix sistema de gestió que les cadenes del llenguatge de programació C, és a dir, una tira de caràcters, amb l'excepció que Btrieve considera que la cadena s'acaba en el primer zero binari que troba.

- logical

  És un tipus de dada que pot ser d'1 o 2 bytes i que Btrieve classifica com si fossin string, és a dir, segons l'ordre lexicogràfic. Aquestes dades es poden utilitzar per a emmagatzemar els dos valors lògics, però també poden tenir altres utilitats.

Per als segments de clau de tipus cadena (string, lstring, zstring, logical), Btrieve permet definir una seqüència d'ordenació alternativa al codi ASCII. Hi pot haver uns segments de clau de tipus cadena classificats segons la seqüència estàndard ASCII i uns altres segments classificats segons una seqüència alternativa (les dues classificacions poden coexistir en segments d'una mateixa clau). Ara bé, només hi pot haver una seqüència alternativa en un fitxer.

b) De tipus numèric

- integer

  Correspon al tipus enter amb signe que ha d'ocupar un nombre parell de bytes (valor definible pel creador del fitxer). Si té 2 bytes es correspon amb el tipus de dades int del llenguatge C (en compiladors de 16 bits). Si té 4 bytes es correspon amb el tipus de dades long del llenguatge C (en compiladors de 16 bits).

- float

  Correspon al tipus real de simple i doble precisió (d'acord amb el nombre de bytes definible pel creador del fitxer) segons l'estàndard IEEE.

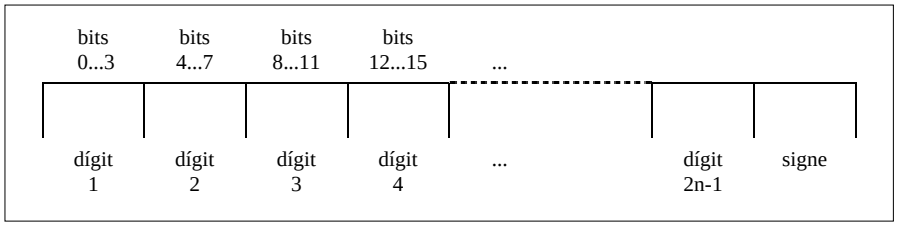
  El format intern d'un valor flotant de 4 bytes (que es correspon amb el tipus de dades float del llenguatge C) consisteix en una mantissa de 23 bits, un exponent de 8 bits desplaçat en 127 i un bit de signe.

  El format intern d'un valor flotant de 8 bytes (que es correspon amb el tipus de dades double del llenguatge C) té una mantissa de 52 bits, un exponent d'11 bits desplaçat en 1023 i un bit de signe.

- decimal

  Correspon al tipus decimal empaquetat, amb dos dígits per byte. La figura 9 ens mostra la representació interna d'un valor de n octets i, per tant, 2n-1 dígits.

Figura 9. Representació del tipus decimal empaquetat



El mig byte de signe tindrà el valor hexadecimal F o C, si el nombre és positiu, i el valor D, si és negatiu. El punt decimal és implícit i tots els valors d'aquest tipus hauran de tenir el mateix nombre de dígits decimals perquè Btrieve pugui classificar-los correctament.

- money

  La representació interna d'aquest valor és idèntica a la del tipus decimal.

- bfloat

  Correspon als nombres reals de simple i doble precisió tal com els tracta el llenguatge Basic de Microsoft.

  El format intern d'un valor flotant de 4 bytes consisteix en una mantissa de 23 bits, un exponent de 8 bits desplaçat en 128 i un bit de signe. El format intern d'un valor flotant de 8 bytes és idèntic, però té una mantissa de 55 bits.

- numerical

  Correspon a valors numèrics emmagatzemats com cadenes; es justifiquen per la dreta i s'omplen amb zeros per l'esquerra. Cada dígit ocupa un byte. El byte situat més a la dreta porta implícitament el signe del valor representat pel nombre, segons mostra la taula 5.

Taula 5. Caràcters que representen el darrer dígit d'un valor numerical en Btrieve, segons el valor numerical sigui positiu o negatiu.

Valor del darrer dígit	1	2	3	4	5	6	7	8	9	0
Positiu	A	B	C	D	E	F	G	H	I	{
Negatiu	J	K	L	M	N	O	P	Q	R	}

Així, per exemple, el valor -3407 es representaria com 340P, doncs en ser negatiu, el dígit 7 es representa amb el caràcter P.

Per a valors positius es permet que el dígit de l'extrem dret mantingui el dígit corresponent (1, 2, ..., 9, 0) enlloc dels valors indicats a la taula 5 (A, B, ... , I, {).

- unsigned

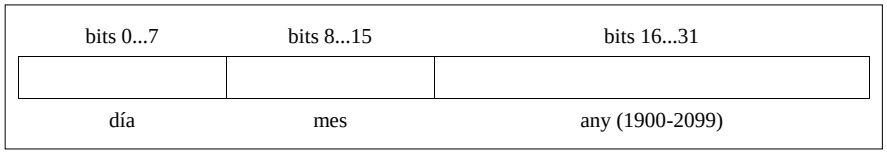
  Correspon als valors enters sense signe que han d'ocupar un nombre parell de bytes. Si té 2 bytes es correspon amb l'unsigned int del llenguatge C. Si té 4 bytes es correspon amb l'unsigned long del llenguatge C.

c) De tipus temporal

- date

  És una dada de 4 bytes que permet gestionar el tipus data (inexistent en el llenguatge C). Té el format que mostra la figura 10.

Figura 10. Representació del tipus de dada date de Btrieve

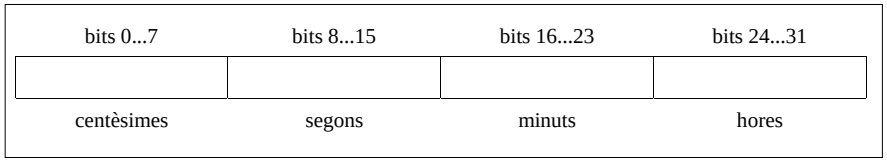


  Els valors dia i mes es tracten en format binari d'un byte cadascun. El valor any es tracta com un binari de 2 bytes que representa el numeral complet de l'any (és a dir, no hi ha efecte 2000!).

- time

  És una dada de 4 bytes que permet gestionar el tipus hora (inexistent en el llenguatge C). Té el format que mostra la figura 11.

Figura 11. Representació del tipus de dada time de Btrieve



  Tots els valors es tracten en format binari d'un byte.

Estructura física dels fitxers

Anem a veure com estan organitzats físicament els fitxers Btrieve. Per a desenvolupar aplicacions informàtiques mitjançant la utilització de l'SGF Btrieve, els coneixements que presentarem en aquest apartat no són necessaris. Ara bé, sí que són imprescindibles per a definir de manera adequada alguns paràmetres en el moment de crear els fitxers Btrieve.

1) Pàgines

Un fitxer Btrieve és format internament per pàgines. Hi ha sempre una pàgina de capçalera, pàgines d'índexs i pàgines de dades.

Una pàgina és la unitat de transferència entre el disc i la memòria intermèdia. La grandària de la pàgina ha de ser múltiple de 512 bytes i no pot ser més gran de 4.096 bytes. (!!)

La pàgina de capçalera conté la informació sobre la grandària i les característiques del fitxer.

Les pàgines de dades poden contenir un o diversos registres, depenent de la longitud de registre definida en crear el fitxer. La part fixa d'un registre no pot estar mai en diferents pàgines.

Les pàgines d'índexs contenen els valors de clau per a accedir als registres del fitxer.

Btrieve manté tots els índexs dels registres en forma d'arbres B. Existeix un arbre B diferent per a cada clau definida en el fitxer.

Btrieve té expansió dinàmica, és a dir, va assignant espai (pàgina a pàgina) de disc a mesura que el va necessitant. Una vegada que s'assigna espai a un fitxer, aquest queda assignat per a tota la vida del fitxer.

Sempre que s'esborra un registre, l'espai que ocupava es col·loca en una llista d'espai lliure que Btrieve utilitzarà per a insercions de registres posteriors.

2) Longitud de registre

Cal distingir dos tipus de longitud de registre: lògica i física.

En fitxers amb registres de longitud variable, el terme longitud de registre lògic s'aplica sempre a la part fixa del fitxer.

Entenem per longitud de registre físic el nombre real de bytes que Btrieve necessita per a emmagatzemar cada registre. Aquesta longitud es calcula afegint, a la longitud de registre lògic, 8 bytes per a cada clau que permeti duplicats i 4 bytes si el fitxer admet registres de longitud variable.

3) Grandària òptima de pàgina

Btrieve emmagatzema tants registres com pot a cada pàgina de dades del fitxer. Sis bytes de cada pàgina s'utilitzen per a informació de gestió del fitxer.

Com que la part fixa d'un registre no pot estar a cavall entre pàgines de dades, pot quedar espai desaprofitat en el fitxer si la grandària de pàgina

menys 6 no és un múltiple exacte de la longitud física del registre. Per optimitzar l'ocupació del disc ha d'escollir-se una grandària de pàgina que pugui emmagatzemar els registres desaprofitant la menor quantitat d'espai. Grandàries menors de pàgina acostumen a donar un rendiment més gran.

4) Compressió de pàgines d'índex

Cada vegada que una pàgina d'índex s'omple, Btrieve en crea automàticament una altra i divideix els valors de la pàgina plena entre les dues noves. Ara bé, és possible executar Btrieve amb l'opció de "compressió de pàgines d'índex", cosa que provoca que no es creï una nova pàgina si hi ha altres pàgines no plenes; en aquest cas mou valors de la pàgina plena a les no plenes.

Treballar amb "compressió de pàgines d'índex" té avantatges i inconvenients. Com a avantatge cal considerar un millor aprofitament general del disc. Com a inconvenient cal saber que Btrieve haurà d'examinar més pàgines del fitxer i, per tant, haurà de fer més accessos al disc.

Cal tenir present que executar Btrieve amb l'opció de compressió en alguna ocasió no implica que sempre s'hagi d'executar amb aquesta opció.

5) Estimació de la grandària de fitxer

És important, en treballar amb fitxers, tenir la possibilitat d'estimar la grandària de fitxer en funció del nombre de registres que aproximadament contindrà, per tal de poder estimar l'espai de disc necessari.

Les fórmules següents es basen en el cas més pessimista, és a dir, en el màxim espai necessari, i estan calculades per a fitxers que no tinguin registres de longitud variable. Tampoc no consideren el cas de treballar amb l'opció de "compressió d'índexs".

Grandària del fitxer = Nombre total pàgines * Grandària de pàgina

Nombre total pàgines = Nombre pàg. dades + Nombre pàg. índexs + 1

Observem les correspondències següents:

NR = Nombre de registres
LR = Longitud de registre lògic
ND = Nombre de claus amb duplicats

TP = Grandària de pàgina
LC = Longitud de clau
NV = Nombre de valors de clau
NVD = Nombre de valors de clau diferents (si permet duplicats)

Es verifica que:

$$Nre.\ pàgines\ de\ dades = \frac{NR*(LR+8*ND)}{TP-6}$$

Per a claus que admetin duplicats tenim que:

$$Nre.\ pàgines\ d'índex = \frac{NVD*(LC+12)}{TP-12}*2$$

Per a claus que no admetin duplicats tenim que:

$$Nre.\ pàgines\ d'índex = \frac{NV*(LC+8)}{TP-12}*2$$

6) Assignació prèvia d'espai a disc

La velocitat d'accés a un fitxer pot millorar-se si totes les seves dades ocupen un espai contigu en el disc, i encara més quan la grandària del fitxer és important.

En el moment de creació es pot assignar a un fitxer Btrieve un espai en disc de fins a 65.535 pàgines. Perquè l'espai assignat en disc sigui contigu, el disc haurà de disposar evidentment d'aquesta quantitat de bytes contigus. Btrieve reservarà l'espai indicat independentment que sigui contigu o no. Si no hi ha prou espai en disc, no el crearà. Cap altre fitxer podrà utilitzar aquest espai.

2.4.4. Integritat de les dades

En presentar les característiques que aportava l'SGF Btrieve, hem comentat que incorporava controls d'integritat en dos nivells.

a) A nivell d'un fitxer, amb la utilització de preimatges.

El gestor de fitxers pot necessitar fer diferents actualitzacions físiques de pàgines per a processar una única petició lògica de l'aplicació. Per tant, pot ser que un fitxer quedi inconsistent si el funcionament de Btrieve es talla a causa d'error del sistema. Btrieve salva aquesta situació amb la tècnica de preimatges.

La tècnica de preimatges consisteix a mantenir un fitxer temporal associat a cada fitxer Btrieve que estigui obert. Aquest fitxer temporal té el mateix nom que el seu fitxer associat, però no té la mateixa extensió, que és sempre .pre. Aquests fitxers també tenen una àrea de memòria intermèdia associada que anomenarem àrea de memòria preimatge. (!!)

Quan s'executa una operació d'actualització, Btrieve segueix els passos següents:

1) Copia les pàgines que hauran de modificar-se, les quals ha portat a les àrees de memòria intermèdia del fitxer per a la seva modificació però que encara no s'han modificat, a les àrees de memòria preimatge.

2) Guarda els registres actualitzats en les corresponents pàgines de les àrees intermèdies de memòria.

3) Copia les àrees de memòria preimatge als fitxers preimatge del disc.

4) Grava les pàgines actualitzades en el fitxer de dades del disc.

5) Esborra les àrees de memòria preimatge dels fitxers preimatge del disc que s'havien gravat en el tercer pas.

Si les àrees de memòria intermèdia i/o preimatge s'omplen abans d'acabar l'operació, Btrieve grava la informació al disc per alliberar aquestes àrees. La informació torna a gravar-se al disc en acabar l'operació.

És important:

- No utilitzar fitxers que tinguin extensió .pre.
- No utilitzar fitxers Btrieve de diferent extensió però del mateix nom, ja que Btrieve confondrà els fitxers preimatge.
- No esborrar mai un fitxer amb extensió .pre.

El fitxer preimatge només existeix mentre el corresponent fitxer Btrieve és obert. Per tant, davant una caiguda del sistema sense que s'hagi tancat correctament el fitxer Btrieve, el corresponent fitxer preimatge continuarà existint en el disc dur. Quan l'SGF Btrieve obre un fitxer Btrieve per a una sol·licitud de qualsevol aplicació, comprova la no-existència del fitxer preimatge associat. En cas de trobar-lo, comprova el contingut del fitxer Btrieve amb el del fitxer preimatge i soluciona la possible inconsistència de les dades assegurant que:

– o bé la darrera operació d'actualització s'ha efectuat correctament,

– o bé la darrera operació d'actualització no s'ha iniciat.

b) Integritat de les dades a nivell de la base de dades: control de transaccions

Si un conjunt d'operacions de Btrieve es troba entre una crida d'inici de transacció i una de final de transacció, cap d'aquestes operacions quedarà registrada fins que no acabin totes satisfactòriament. Hi pot haver fins a 12 fitxers diferents en una mateixa transacció. (!!)

Btrieve utilitza un fitxer per controlar les transaccions i mantenir un seguiment dels fitxers implicats. El nom d'aquest fitxer el decideix l'usuari en posar en funcionament l'SGF. Quan es torni a posar en marxa l'SGF després d'una caiguda del sistema, caldrà especificar el mateix nom de fitxer de transaccions.

Btrieve busca el fitxer de transaccions per dur a terme qualsevol procediment de recuperació que sigui necessari. En la majoria dels casos, la recuperació de la transacció té lloc quan es tornen a obrir els fitxers implicats en la transacció. No obstant això, si el sistema falla quan Btrieve està processant un "final de transacció", part de la recuperació s'esdevé en el moment de posar en marxa l'SGF. En aquest cas, tots els fitxers implicats en la transacció han de ser en el mateix lloc on eren abans de la caiguda del sistema.

Btrieve utilitza també els fitxers preimatge per preservar la integritat del fitxer en l'execució de transaccions. L'única diferència és que els fitxers preimatge creixen durant la transacció i no solament durant una operació. Per tant, els fitxers preimatge poden necessitar més espai en el disc quan s'utilitzen transaccions en els programes.

2.4.5. Optimització del rendiment

La recuperació automàtica de dades de Btrieve suposa normalment una sobrecàrrega de temps de procés. Com que hi ha circumstàncies en què aquesta sobrecàrrega no és justificable, Btrieve inclou opcions que permeten inhibir la lògica de la recuperació automàtica. Per tant, es pot millorar el rendiment en les operacions d'actualització (mai en les de lectura).

Hi ha dues opcions per a millorar-ne el rendiment:

a) Preimatge dirigida

Per defecte, el fitxer preimatge es crea en el mateix disc lògic en què resideix el fitxer Btrieve. Es pot forçar que tots els fitxers preimatge es creïn en un disc diferent. Aquest disc haurà de tenir una estructura de directoris similar en tots els seus nivells a la que conté el fitxer Btrieve.

Si la preimatge es dirigeix a un disc RAM (disc virtual), la preimatge no produeix cap activitat addicional d'accés al disc. Però llavors Btrieve no podrà fer la recuperació automàtica en cas que el sistema falli durant una operació d'actualització o en qualsevol moment d'una transacció. En aquest cas, els registres es podran recuperar sobre un fitxer seqüencial mitjançant la utilitat RECOVER que proporciona l'SGF Btrieve.

b) Obertura en mode accelerat

Quan un fitxer s'obre en mode accelerat, Btrieve no efectua la sortida física al disc fins que l'àrea de memòria intermèdia corresponent és plena i l'algorisme UMR selecciona una pàgina per a reutilitzar l'espai que ocupa. No escriurà en els fitxers preimatge a menys que l'àrea de memòria preimatge s'ompli al llarg d'una única operació. Tampoc no actualitza l'estructura del directori a mesura que creix físicament el fitxer. En cas que hi hagi un error del sistema, Btrieve no pot recuperar automàticament el fitxer la propera vegada que s'obri. Només es podrà fer servir la utilitat RECOVER.

En entorns multiusuari, obrir un fitxer en mode accelerat suposa un cert tipus de bloqueig que depèn de l'entorn multiusuari en concret. Així:

– L'obertura en mode accelerat sota Btrieve/N per DOS provoca accés exclusiu al fitxer. Ningú més no pot accedir-hi fins que no es tanqui.

– En un sistema en xarxa o multiusuari, es pot obrir un fitxer en mode accelerat des de diversos llocs de treball, però només amb aquest mode d'obertura. Qualsevol intent d'obertura en mode normal mentre el fitxer és obert en mode accelerat provocarà un error de "mode incompatible".

2.4.6. Controls de concurrència

En entorns de xarxa i multiusuari, Btrieve proporciona tres mètodes diferents per a resoldre conflictes entre diferents llocs de treball que intenten actualitzar o esborrar simultàniament els mateixos registres. Aquests mètodes poden utilitzar-se individualment o combinats en un mateix programa i són:

- Control de transaccions
- Mètode passiu
- Bloqueig de registres

Anem a veure'n cadascun d'ells amb deteniment. Cal tenir clar, però, que per dur-los a la pràctica caldrà aplicar de manera adequada les operacions que corresponguin d'entre el conjunt d'operacions facilitat per Btrieve.

Trobareu informació detallada sobre les operacions facilitades per l'SGF Btrieve en el contingut "SGF Btrieve 5.10a. Operacions i codis d'error" de la web d'aquest crèdit.

a) Control de transaccions

Sempre que un programa accedeix a un fitxer dins una transacció, Btrieve bloqueja el fitxer per aquest lloc de treball fins que la transacció finalitzi o avorti. Es pot llegir el fitxer des d'altres llocs de treball, sempre que la lectura no provingui d'una transacció. No obstant això, ningú més no pot accedir al fitxer des d'una transacció, ja que qualsevol accés des de dins d'una transacció implica l'intent de bloqueig, i no es pot bloquejar un fitxer que ja ho està.

Btrieve executa normalment bloqueigs "amb espera". Si dos llocs intenten accedir al mateix fitxer des d'una transacció, el segon lloc s'esperarà (programa aturat) fins que el primer hagi completat la transacció. No obstant això, si s'obre una transacció amb el codi d'operació 219 en lloc del codi 19, Btrieve executarà bloqueigs "sense espera" durant la transacció. Si dos llocs intenten accedir al mateix fitxer des d'una transacció, el segon a intentar-ho rebrà un codi d'error que indicarà el bloqueig del registre (el programa no queda aturat). En apartats posteriors veurem detingudament la gestió de bloqueigs.

Trobareu informació detallada sobre les operacions 19 i 219 facilitades per l'SGF Btrieve en el contingut "SGF Btrieve 5.10a. Operacions i codis d'error" de la web d'aquest crèdit.

Com que una transacció bloqueja temporalment un fitxer, és important que els programes que utilitzen transaccions segueixin aquestes directrius:

1) Un programa no hauria d'efectuar mai una entrada de dades per teclat dins una transacció, ja que l'usuari del programa pot ser que no respongui la sol·licitud d'entrada i la transacció quedi aturada durant molta estona, la qual cosa provocaria que cap altre lloc de treball pogués actualitzar els fitxers als quals s'ha accedit durant la transacció mentre aquesta no finalitzés.

2) Per a evitar "abraçades mortals", totes les transaccions que accedeixin al mateix grup de fitxers haurien d'executar-se amb l'opció "sense espera" o bé haurien de fer-ho en el mateix ordre. En cas contrari, es pot arribar a una situació en què un lloc de treball vulgui accedir a un fitxer que un altre ja ha bloquejat.

La taula 6 mostra la interacció entre dos llocs de treball executant el mateix programa, que llegeix i actualitza dos fitxers en una transacció "amb espera". Els passos estan numerats segons l'ordre en què s'executen les operacions.

Taula 6. Interacció entre dos llocs de treball que intenten actualitzar dos fitxers Btrieve

Lloc 1 - Aplicació X	SGF Btrieve	Lloc 2 - Aplicació X
1) Obrir fitxer 1		
		2) Obrir fitxer 1
3) Obrir fitxer 2		
		4) Obrir fitxer 2
5) Inici transacció		
		6) Inici transacció
7) Llegir fitxer 1		
		8) Llegir fitxer 1
	9) Espera en lloc 2 ja que el lloc 1 ha bloquejat el fitxer 1	
10) Llegir fitxer 2		
11) Actualitzar fitxer 1		
12) Actualitzar fitxer 2		
13) Final transacció		
	14) Final de lectura pendent en lloc 2	
		15) Llegir fitxer 2
		16) Actualitzar fitxer 1
		17) Actualitzar fitxer 2
		18) Final transacció

La taula 7 mostra com dos llocs de treball en una xarxa poden arribar a una "abraçada mortal". Els llocs estan executant dos programes diferents que accedeixen als mateixos fitxers en diferent ordre dins de transaccions "amb espera". Els passos estan numerats segons l'ordre en què s'executen les operacions.

b) Mètode passiu

El mètode passiu consisteix, com el seu nom indica, a "ser passiu" i no aplicar control de transaccions ni bloqueigs. En aquest cas, caldrà controlar les possibles situacions anòmales que es puguin produir, com ara la que s'observa en la taula 8.

El mètode passiu pot combinar-se de manera eficaç amb el control per transaccions. Si s'accedeix a un fitxer des d'una transacció, els altres llocs de treball no poden actualitzar-lo ni accedir-hi des de les seves

pròpies transaccions. Si té lloc un intent d'accés, Btrieve espera fins que finalitzi la transacció que té el fitxer bloquejat (si la transacció és "amb espera") o retorna un codi d'error (si la transacció és "sense espera"). Qualsevol lloc pot, d'altra banda, llegir un fitxer si les lectures s'efectuen fora d'una transacció.

Taula 7. Interacció entre dos llocs de treball que intenten actualitzar dos fitxers Btrieve

Lloc 1 - Aplicació X	SGF Btrieve	Lloc 2 - Aplicació Y
1) Obrir fitxer 1		
2) Obrir fitxer 2		
		3) Obrir fitxer 1
		4) Obrir fitxer 2
5) Inici transacció		
		6) Inici transacció
7) Llegir fitxer 1		
		8) Llegir fitxer 2
9) Llegir fitxer 2		
	10) Espera en lloc 1 ja que el lloc 2 ha bloquejat el fitxer 2	
		11) Llegir fitxer 1
	12) Espera en lloc 2 ja que el lloc 1 ha bloquejat el fitxer 1	

Taula 8. Situació anòmala que es pot produir entre programes que actuen amb mètode passiu.

Lloc 1 - Aplicació X	SGF Btrieve	Lloc 2 - Aplicació Y
1) Obrir fitxer 1		
		2) Obrir fitxer 1
3) Llegir registre amb clau A		
		4) Llegir registre amb clau A
5) Actualitzar registre amb clau A		
		6) Actualitzar registre amb clau A
	7) L'SGF retorna al lloc 2 un error (codi 80) ja que es vol actualitzar un registre el contingut del qual ha canviat des del moment en què es va produir la lectura	
		8) Reintentar el pas (4)

La taula 9 mostra la interacció entre un programa d'aplicació que utilitza un control de transacció amb espera i un altre programa d'aplicació amb utilització de mètode passiu.

Taula 9. Interacció entre programa amb control de transaccions i programa amb mètode passiu.

Lloc 1 - Aplicació X	SGF Btrieve	Lloc 2 - Aplicació Y
1) Obrir fitxer 1		
2) Obrir fitxer 2		
		3) Obrir fitxer 1
4) Inici transacció		
5) Llegir en fitxer 1 el registre de clau A		
		6) Llegir en fitxer 1 el registre de clau B
		7) Actualitzar en fitxer 1 el registre de clau B
	8) Espera en lloc 2 ja que el lloc 1 ha bloquejat el fitxer 1	
9) Llegir en fitxer 2 el registre de clau A		
10) Actualitzar en fitxer 1 el registre de clau A		
11) Actualitzar en fitxer 2 el registre de clau A		
12) Final transacció		
	13) Final de l'actualització pendent en el lloc 2	

c) Bloqueig de registres

Quan un programa accedeix a un registre d'un fitxer mitjançant una instrucció de lectura, pot especificar que el registre quedi bloquejat. Altres llocs de treball podran llegir el registre, sempre que les lectures s'executin sense opcions de bloqueig. Cap altre lloc podrà bloquejar el registre fins que sigui desbloquejat pel lloc que el va bloquejar per primera vegada.

En general, es permeten bloqueigs amb espera i bloqueigs sense espera.

Una instrucció de lectura amb opció de bloqueig amb espera provoca que l'SGF intenti bloquejar el registre mitjançant el programa d'aplicació que executa la instrucció. Si el registre a accedir està bloquejat, el programa d'aplicació es queda en espera indefinida fins que el registre sigui desbloquejat pel lloc de treball que l'havia bloquejat.

Una instrucció de lectura amb opció de bloqueig sense espera provoca que l'SGF intenti bloquejar el registre mitjançant el programa d'aplicació que executa la instrucció. Si el registre a accedir està bloquejat, el programa d'aplicació rep la corresponent notificació de registre ja bloquejat.

En treballar en entorns en xarxa i en entorns multiusuari, els programes d'aplicació han d'estar convenientment dissenyats per a fer front a les situacions de bloqueig que es puguin produir.

Ens trobem dos tipus de bloqueig segons la quantitat de registres que es puguin mantenir bloquejats de manera simultània en un fitxer: bloqueig individual i bloqueig múltiple.

1) Bloqueig individual

En utilitzar el bloqueig individual sobre un registre d'un fitxer, s'impedeix que hi pugui haver més d'un registre bloquejat simultàniament en el mateix fitxer. (!!)

Una vegada un lloc de treball ha bloquejat individualment amb èxit un registre, el bloqueig es manté fins que tingui lloc algun dels fets següents:

- El lloc de treball actualitza o esborra el registre bloquejat.
- El lloc de treball bloqueja un altre registre del mateix fitxer.
- El lloc de treball executa una operació de desbloqueig sobre el fitxer (no cal dir el registre sobre el qual s'aplica, ja que només hi pot haver un registre bloquejat en un moment donat).
- El lloc de treball tanca el fitxer.
- El lloc de treball executa una operació de restauració per tancar tots els fitxers que tingui oberts.

2) Bloqueig múltiple

En utilitzar el bloqueig múltiple sobre un registre d'un fitxer es continua bloquejant un únic registre, però el registre bloquejat no s'allibera automàticament en executar una nova lectura amb bloqueig múltiple ni en actualitzar el registre bloquejat. (!!)

Els registres bloquejats es mantenen fins que té lloc algun dels fets que s'esmenten a continuació:

- El lloc de treball que ha bloquejat els registres executa una operació de desbloqueig, que pot ser per a un sol registre o per a tots els registres bloquejats del fitxer.

- El lloc de treball executa una operació de restauració per tancar tots els fitxers que té oberts. Es desbloquegen tots els registres bloquejats.

- El lloc de treball tanca el fitxer. Es desbloquegen tots els registres bloquejats.

- El lloc de treball esborra el registre. Es desbloqueja el registre esborrat.

2.4.7. Com utilitzar l'SGF Btrieve sobre DOS i amb llenguatge C

Els coneixements adquirits fins al moment sobre l'SGF Btrieve són genèrics. Ara bé, si volem practicar amb l'SGF Btrieve, caldrà conèixer com hi podem treballar en funció del sistema operatiu i del llenguatge de programació.

En aquest apartat veurem els fitxers necessaris per a treballar-hi sobre MS-DOS i desenvolupar programes amb llenguatges d'alt nivell.

Fitxer `btrieve.exe`

Aquest és el fitxer executable de l'SGF Btrieve per al sistema operatiu MS-DOS. Ha d'estar carregat (resident) perquè qualsevol programa d'aplicació pugui efectuar operacions sobre fitxers Btrieve.

Btrieve es carrega executant, des del sistema, el comandament següent:

```
BTRIEVE[/M:m][/P:p][/T:t][/I:i][/C:][/B:b][/F:f][/E][/L:l]
```

Btrieve queda resident fins que es descarrega amb l'ajuda de la utilitat `butil`.

La taula 10 mostra la funcionalitat de les diferents opcions d'arrencada.

Taula 10. Funcionalitats de les diferents opcions d'arrencada de l'SGF Btrieve

`/M:m`	Grandària màxima `m` en KB per a l'àrea de memòria intermèdia que pot utilitzar Btrieve. El valor per defecte és 32 KB. Pot valer de 9 KB a 64 KB. Si existeix memòria expandida, Btrieve la utilitzarà per a les àrees de memòria intermèdia i assignarà sempre 64 KB sense considerar `/M`.
`/P:p`	Grandària màxima `p` de pàgina, en bytes, per a qualsevol fitxer Btrieve al qual es vulgui accedir. El valor per defecte és 2048. Ha de ser un múltiple de 512 i no més petit de 4096. Com major sigui el valor de `p`, menys àrees parcials podrà assignar Btrieve dins la memòria assignada amb `/M`.
`/T:t`	Nom `t` del fitxer de control de transaccions que ha d'utilitzar Btrieve.
`/I:i`	Adreça els fitxers preimatge a un dispositiu diferent d'aquell en el qual resideixen els fitxers Btrieve.
`/C`	Comprimeix índexs.

`/B:b`	Quantitat total `b` de memòria en KB que pot utilitzar Btrieve per a àrees preimatge. El valor per defecte és de 16 KB. Pot valer d'1 KB a 16 KB. Aquestes àrees sempre són en la memòria principal, encara que hi hagi memòria expandida.
`/F:f`	Nombre màxim `f` de fitxers Btrieve que hi pot haver simultàniament oberts. El valor per defecte és 20 i pot especificar-se un valor màxim de 255.
`/E`	Evita la utilització de la memòria expandida.
`/L:l`	Indica el nombre màxim simultani de bloqueigs actius de registre, especificant el nombre de posicions que tindrà la taula de bloqueigs construïda per l'SGF en temps de càrrega. El valor per defecte és 20 i pot especificar-se un valor màxim de 255.

Fitxer `butil.exe`

Aquest és el fitxer executable corresponent a un conjunt d'utilitats que Btrieve aporta, en el sistema operatiu MS-DOS, per a gestionar fitxers Btrieve sense necessitat de fer un programa d'aplicació. La seva evolució lògica seria una consola gràfica d'administració per l'SGF.

A continuació, tenim les diferents opcions d'execució d'aquesta utilitat. No és necessari tenir-les memoritzades ni tampoc no cal tenir el manual d'utilització al costat, ja que amb una simple execució sense cap opció se'ns visualitzen:

```
C:\>butil
Btrieve Utilities Version 5.12
Copyright 1982 - 1990, Novell, Inc.  All Rights Reserved.
 Usage is BUTIL -CLONE <OUTPUT-FILE> <SOURCE-FILE> [-O<OWNER1>]
 Usage is BUTIL -COPY <INPUT-FILE> <OUTPUT-FILE> [-O<OWNER1> [-O<OWNER2>]]
 Usage is BUTIL -CREATE <FILE-NAME> <DESCRIPTION FILE>
 Usage is BUTIL -DROP <BTRIEVE-FILE> <KEY NUMBER> [-O<OWNER>]
 Usage is BUTIL -EXTEND <BTRIEVE-FILE> <EXTENSION-FILE> [<IMMEDIATE(Y/N)>]
     [-O<OWNER>]
 Usage is BUTIL -INDEX <BTRIEVE-FILE> <INDEX-FILE> <DESCRIPTION-FILE>
     [-O<OWNER>]
 Usage is BUTIL -LOAD <INPUT-FILE> <BTRIEVE-FILE> [-O<OWNER>]
 Usage is BUTIL -RECOVER <BTRIEVE-FILE> <OUTPUT-FILE> [-O<OWNER>]
 Usage is BUTIL -RESET [<STATION NAME> | <USER NAME> | <PROCESS ID>]
 Usage is BUTIL -SAVE <BTRIEVE-FILE> <OUTPUT-FILE>
     [<INDEX(Y/N)> <INDEX-FILE> | <KEY NUMBER>] [-O<OWNER>]
 Usage is BUTIL -SINDEX <BTRIEVE-FILE> <DESCRIPTION-FILE> [-O<OWNER>]
 Usage is BUTIL -STAT <FILE-NAME> [-O<OWNER>]
 Usage is BUTIL -STOP
 Usage is BUTIL -VER
```

Podem observar que hi ha 14 opcions diferents d'utilització de la utilitat `butil.exe`. Vegem, a continuació, una breu explicació de cadascuna.

```
BUTIL -VER
```

Dóna informació sobre la versió carregada de l'SGF Btrieve o ens comunica que l'SGF no està carregat.

```
BUTIL -STOP
```

Descarrega l'SGF Btrieve de la memòria RAM.

```
BUTIL -CREATE <Fitxer_Btrieve> <Fitxer_Descripció>
```

Genera un fitxer Btrieve buit, amb les característiques que s'especifiquin en el fitxer descripció el qual ha de ser un fitxer de tipus text i ha de contenir una sèrie d'elements, cadascun dels quals va precedit d'una paraula clau, en minúscules, que el defineix.

L'ordre d'especificació d'elements és el següent: una part inicial (capçalera) que conté els apartats descrits a la taula 11 i una part per a cada segment de clau amb els continguts descrits a la taula 12.

Taula 11. Descripció dels possibles apartats de la capçalera d'un fitxer de descripció per als fitxers Btrieve

`record=`	Longitud de registre lògic.
`variable=`	Ha de posar-se `y` si pot contenir registres de longitud variable; en cas contrari, ha de posar-se `n`.
`compression=`	Només s'hi pot posar si el fitxer accepta registres de longitud variable; ha de posar-se `y` si es vol compressió de blancs a la dreta de la part variable del registre, la qual cosa estalvia espai en disc.
`key=`	Nombre de claus d'accés que es definiran.
`page=`	Grandària de pàgina del fitxer, en bytes.
`allocation=`	Només cal posar-lo si es vol assignació prèvia d'espai al disc; cal fer-hi constar el nombre de pàgines assignades inicialment, que pot ser fins a 65.535.

Taula 12. Descripció dels possibles apartats per a cada segment de clau en un fitxer de descripció per als fitxers Btrieve

`position=`	Byte del registre en el qual comença el segment; els bytes del registre es numeren a partir d'1 (no de zero).
`Length=`	Longitud, en bytes, del segment de clau.
`duplicates=`	Ha de posar-se `y` si pot admetre duplicats per aquest segment de clau; en cas contrari, ha de posar-se `n`.
`modifiable=`	Ha de posar-se `y` si els valors que prendre aquest segment es podran modificar; en cas contrari, ha de posar-se `n`.
`type=`	Ha de posar-se el tipus de dada amb què Btrieve ha d'interpretar el segment de clau (`zstring`, `logical`...).
`descending=`	Aquest atribut no és obligatori, però cal posar-hi `y` si es vol que Btrieve classifiqui en ordre descendent.
`alternate=`	Només té significat per a segments de claus literals (però s'hi ha de posar sempre), i ha de contenir `y` si es vol que Btrieve classifiqui aquest índex amb una seqüència alternativa al codi ASCII; en cas contrari, ha de posar-se `n`.
`null=`	Ha de posar-se `y` si es vol definir un valor nul per a aquest segment de clau; en cas contrari, ha de posar-se `n`.
`value=`	Només s'hi ha de fer constar si s'ha posat `y` a l'atribut nul; en aquest cas, ha de contenir el codi ASCII del valor que s'ha de considerar com a valor nul.
`segment=`	Ha de posar-se `y` si a continuació ve un altre segment corresponent a la mateixa clau; en cas contrari, ha de posar-se `n`.

Tots els segments de clau s'han de posar un darrere l'altre, en l'ordre que formen la clau. D'aquesta manera, Btrieve interpreta, en crear el fitxer, l'ordre de les claus i les numera començant per la clau zero.

En el cas que un o diversos segments de clau literals hagin de classificar-se mitjançant una seqüència alternativa, encara ha d'afegir-se al fitxer de descripció una part final (peu) que conté els valors de la taula 13.

Taula 13. Descripció dels possibles apartats del peu d'un fitxer de descripció per als fitxers Btrieve

name=	nom del fitxer que conté la definició de la seqüència alternativa.

El fitxer de seqüència alternativa és un fitxer de text que ha de tenir el contingut que mostra la taula 14.

Taula 14. Contingut del fitxer de seqüència alternativa a emprar per un fitxer Btrieve.

Posició	Longitud	Contingut
0	1	Byte d'identificació que ha de contenir sempre el valor hexadecimal AC (172 decimal).
1	8	Un nombre de 8 bytes que identifica de forma unívoca per part de Btrieve la seqüència alternativa de classificació, ja que diferents fitxers poden utilitzar seqüències diferents.
9	256	Taula que conté a cada posició p el nou valor de classificació del caràcter que ocupa el lloc p a la taula ASCII. És a dir, sabem que el caràcter 'A' ocupa el lloc 65 decimal de la taula ASCII. Si volem que ocupi un altre lloc, per exemple el lloc 80 decimal, caldrà que a la posició 65 de la taula hi hagi el caràcter ASCII corresponent al valor 80 decimal.

Una vegada el fitxer Btrieve sigui generat, el fitxer de seqüència alternativa pot desaparèixer, la qual cosa indica que la definició de la seqüència alternativa queda enregistrada en el mateix fitxer Btrieve.

Un fitxer Btrieve es pot crear amb la utilitat `butil.exe` o a través d'un programa amb l'execució de l'operació de creació corresponent. Si us ha semblat embolicat el procés per a crear el fitxer, és a dir, si us ha semblat que s'ha de donar molta informació, penseu que per a crear el fitxer des d'un programa també s'haurà de comunicar tota aquesta informació a l'SGF, cosa que s'haurà de fer mitjançant la utilització de variables. Per tant, potser encara és més complicat crear un fitxer Btrieve des d'un programa que des de l'eina `butil.exe`.

D'altra banda, cal tenir en compte que els fitxers Btrieve que es gestionen en una organització acostumen a crear-se una única vegada quan es posa en marxa l'aplicació. Per tant, quin sentit té assignar càrrega de programació a la creació de fitxers Btrieve (per a una única vegada) quan disposem d'una utilitat que ens els crea sense necessitat de programar?

```
BUTIL -STAT <Fitxer_Btrieve> [-O<Paraula_pas>]
```

Mostra les característiques de definició d'un fitxer Btrieve així com informació estadística del seu contingut.

De la sintaxi d'execució d'aquesta opció es dedueix que els fitxers Btrieve poden tenir una paraula de pas assignada. Ja ho havíem fet notar en la descripció de les característiques de l'SGF Btrieve. L'assignació d'una determinada paraula així com l'eliminació no es poden efectuar amb l'eina `butil.exe`, sinó que s'han de fer des d'un programa amb les operacions corresponents.

Quan es vulgui accedir a un fitxer Btrieve amb paraula de pas a través de la utilitat `butil.exe`, caldrà comunicar la paraula en el moment d'executar la instrucció `butil.exe` corresponent. L'eina `butil.exe` menysprea els blancs a la dreta de la paraula. Hi ha la possibilitat d'introduir un asterisc com a primer caràcter. En aquest cas, `butil.exe` demana la introducció de la paraula de pas i no menysprea els caràcters blancs que s'introdueixin a la dreta de la paraula.

Si el comandament butil.exe a executar conté el nom de dos fitxers Btrieve, cal introduir dues paraules de pas, una per a cada fitxer. Si només el segon fitxer té paraula de pas, `butil.exe` ignorarà el valor introduït com a paraula de pas pel primer fitxer. En cas que hi hagi dos fitxers es poden utilitzar dos asteriscos.

```
BUTIL -CLONE <Fitxer_sortida> <Fitxer_font> [-O<Paraula>]
```

Crea una còpia buida del fitxer font Btrieve.

```
BUTIL -COPY <Fit_entrada> <Fit_sortida>    [-O<Paraula_entr> [-O<Paraula_sort>]]
```

Copia tots els registres del fitxer d'entrada en el fitxer de sortida, que ha d'existir i pot ser que no estigui buit. Es pot utilitzar l'opció `COPY` quan cal canviar les característiques de clau d'un fitxer.

```
BUTIL -SINDEX <Fitxer> <Fitxer_descripció> [-O<Paraula>]
```

S'utilitza per a afegir índexs (suplementaris) als que es van definir en el moment de la creació del fitxer (permanents).

Btrieve classifica els valors duplicats en els índexs permanents segons l'ordre cronològic d'incorporació al fitxer. En canvi, en els suplementaris, els valors duplicats s'indexen segons la seva ubicació física en el fitxer.

El fitxer de descripció ha de contenir la definició de la clau, de la mateixa manera que es defineix una clau en el moment de la creació d'un fitxer Btrieve.

Les claus que es creen amb els índexs suplementaris van numerades, d'acord amb l'ordre de les que ja existien en el fitxer.

Aquests índexs s'actualitzen automàticament, com els índexs permanents. La utilitat `-STAT` indica les claus suplementàries amb una `S` després del número de clau.

```
BUTIL -DROP <Fitxer> <Número_clau> [-O<Paraula>]
```

Permet eliminar un índex suplementari d'un fitxer Btrieve.

Si s'han definit índexs suplementaris després del que es vol eliminar, que, per tant, tenen un número de clau més gran, Btrieve disminueix en una unitat aquells números de clau quan s'executi l'opció d'eliminació de l'índex suplementari.

```
BUTIL -INDEX <Fitxer> <Fitxer_índex> <Fitxer_descripció> [-O<Paraula>]
```

Permet indexar un fitxer ja existent a través d'una nova clau que no es va definir en el moment de crear el fitxer. Aquest índex no s'actualitza automàticament. Més que res, serveix per a utilitzar posteriorment l'opció `-SAVE` amb els registres indexats per una determinada clau que no existeix com a índex permanent ni com a suplementari.

El fitxer de descripció ha de contenir la definició de la clau, de la mateixa manera que es defineix una clau en el moment de la creació d'un fitxer Btrieve.

```
BUTIL -SAVE <Fitxer_Btrieve> <Fitxer_sortida> [<(Y/N)> <Fitxer_índex> | <Número_clau>]
[-O<Paraula>]
```

Permet copiar registres d'un fitxer Btrieve en un fitxer de text.

Per a cada registre que llegeix del fitxer, Btrieve genera un registre en el fitxer de text que conté, primer, la longitud del registre seguida d'una coma, el registre en qüestió i, finalment, un retorn de carro i un salt de línia. No fa cap conversió sobre el tipus de dades dels camps del registre.

Si es vol omplir el fitxer de text seguint l'ordre d'un índex extern, ha de posar-se `y` després del nom del fitxer de text i afegir a continuació el nom del fitxer d'índex extern. Si es vol seguir l'ordre d'una clau diferent de la

clau `0` del fitxer, ha de posar-se `n` i a continuació el número de clau que es desitja.

Si quan està copiant els registres no hi ha prou espai, dóna un missatge i permet continuar en un altre disc o finalitzar.

```
BUTIL -LOAD <Fitxer_entrada><Fitxer_Btrieve>[-O<Paraula>]
```

Permet afegir registres des d'un fitxer text en un fitxer Btrieve.

`LOAD` suposa que cada registre en fitxer text comença amb la longitud de registre en ASCII, un separador de caràcter (“,” o “ ”), el registre i un retorn de carro i un salt de línia.

```
BUTIL -EXTEND <Fitxer> <Fitxer_ext>[<(Y/N)>][-O<Paraula>]
```

Crea fitxers repartits en dos discos diferents.

Normalment Btrieve no començarà a utilitzar el fitxer estès fins que el disc del fitxer original sigui ple. Però si es desitja que Btrieve comenci a omplir el fitxer extensió immediatament, cal que es posi `y` després del nom del fitxer extensió.

```
BUTIL -RECOVER <Fitxer> <Fitxer_Sortida> [-O<Paraula>]
```

Permet recuperar dades d'un fitxer Btrieve corrupte a causa d'un error del sistema mentre s'estava accedint al fitxer en accés accelerat o amb preimatge dirigida a un disc RAM.

Guarda les dades en un fitxer text compatible amb `LOAD` i `SAVE`.

```
BUTIL -RESET [<Nom_Estació> | <Nom_Usuari> | <Id_Procés>]
```

Permet alliberar els recursos assignats per Btrieve en un lloc de treball. Allibera tots els bloqueigs, avorta les transaccions pendents i tanca els fitxers oberts pel lloc de treball.

En Btrieve monousuari no es requereix el paràmetre identificador del lloc de treball.

Rutines d'interfície amb llenguatges d'alt nivell

Els programes `btrieve.exe` i `butil.exe` són fitxers executables en el sistema operatiu DOS. En altres sistemes operatius hi ha els programes

equivalents. No hem vist encara, però, com podem codificar programes que efectuïn crides a l'SGF.

En parlar de les característiques de l'SGF Btrieve, comentàvem que es podia codificar en diferents llenguatges de programació. Això s'aconsegueix mitjançant la utilització d'una rutina d'interfície específica per a cada llenguatge de programació.

El programari que acompanya Btrieve incorpora rutines d'interfície per als principals compiladors dels llenguatges de programació d'alt nivell més importants. Fixeu-vos bé que la rutina d'interfície s'ha associat no únicament al llenguatge de programació, sinó també al compilador utilitzat. A més, es poden efectuar crides al SGF Btrieve en llenguatge assemblador i, fins i tot, escriure una rutina d'interfície pròpia.

L'SGF Btrieve facilita rutines d'interfície per als principals compiladors de llenguatge C. Aquestes rutines consisteixen en un fitxer codificat en llenguatge C diferent per a cada compilador. Suposem que el fitxer que conté la rutina s'anomena rutina.c (en realitat, el nom és diferent per a cada compilador comercial).

Hi ha dues maneres d'utilitzar aquest fitxer:

- Amb la directiva de precompilació `#include` en els programes font.
- Definint el corresponent rutina.h que contingui les declaracions necessàries, efectuant `#include` d'aquest fitxer en els fitxers fonts i realitzant l'enllaç dels programes objectes generats i el corresponent rutina.obj en el moment de generar el fitxer executable.

Podem utilitzar la primera opció si el nostre programa només està format per un fitxer `.c`, però en cas de tenir varis arxius `.c` que precisin accedir a Btrieve, caldrà utilitzar la segona opció.

El fitxer `rutina.c` en qüestió conté la funció `BTRV`, que cal utilitzar en totes les operacions Btrieve executades des d'un programa codificat en llenguatge C. El seu prototipus és:

```
int BTRV(op, fitxer, registre, long_reg, àrea_clau, num_clau)
     int op;               /* codi d'operació */
     char fitxer[128]      /* fitxer intern per a Btrieve */
     char *registre        /* registre */
     int *long_reg         /* longitud de registre */
     char *àrea_clau       /* valor per la clau activa */
     int núm_clau          /* número de la clau activa */
```

Els programes mai no han de modificar el contingut de l'argument `fitxer`, doncs és el fitxer intern per a Btrieve i és on Btrieve manté

informació actualitzada corresponent al fitxer extern (clau activa, registres actual, anterior i següent...).

L'SGF Btrieve pot canviar, després de qualsevol crida a la funció `BTRV`, els valors de `registre`, `long_reg` i `àrea_clau`, ja que són arguments passats per variable o referència. Btrieve mai no canviarà, en canvi, el valor d'`op` i `núm_clau`.

Fixeu-vos que totes les operacions es realitzen amb la crida a una única funció, diferenciant, via argument, l'operació a efectuar.

La funció `BTRV` retorna sempre un valor `int` indicador de la correcta o incorrecta execució de la funció. Un resultat `zero` indica execució correcta. Un resultat diferent de `zero` indica execució errònia i, a més, es correspon amb un codi d'error inclòs en una taula explicativa dels diferents codis d'error. El programador ha d'utilitzar sempre el codi d'error retornat per a provocar, en el programa, l'actuació que correspongui.

Cal fer una observació molt important respecte als arguments `registre` i `àrea_clau`, ja que la funció `BTRV` els porta declarats com a punters `char *`. L'argument `registre` s'utilitza, normalment, per a obtenir el registre cercat i l'argument `àrea_clau` es fa servir, generalment, per a passar a l'SGF el valor de la clau a cercar, però l'SGF ens hi retorna el valor de la clau corresponent al registre trobat. Per tant, sembla que els dos arguments haurien de ser de diferents tipus en funció de l'estructura del fitxer a accedir i, en canvi, són sempre de tipus `char *`. Com és possible això?

Sabem que un `char` ocupa un octet i que qualsevol altre tipus de dada és un múltiple d'un octet. Per exemple, l'estructura `t_persona` que estem utilitzant contínuament ocupa 74 bytes. Podríem dir que ocupa 74 caràcters, oi? És a dir, Btrieve, quan cerqui un registre a un fitxer de `t_persona`, sap que ha d'anar a cercar 74 bytes i retornar-los en l'argument `registre` que li passem per referència. Per això, l'argument de la funció `BTRV` està declarat com a `char *`, ja que nosaltres li passarem la direcció d'una variable del tipus que sigui i Btrieve prendrà aquesta adreça com un lloc on pot deixar un enfilall de caràcters.

2.4.8. Operacions Btrieve i errors retornats

Per poder dissenyar programes en llenguatge C que gestionin arxius Btrieve ja només ens manca conèixer les diferents operacions que facilita l'SGF Btrieve així com els codis d'error que retornen aquestes operacions.

Trobareu informació detallada sobre les operacions facilitades per l'SGF Btrieve així com els codis d'error que retornen dites funcions, en el contingut "SGF Btrieve 5.10a. Operacions i codis d'error" de la web d'aquest crèdit.

2.4.9. Exemple de gestió de fitxers Btrieve

La pràctica de gestió de fitxers Btrieve només la podrem dur a terme en MS-DOS i via programes desenvolupats en Turbo C, doncs només disposem dels programes `btrieve.exe` i `butil.exe` per a MS-DOS i del fitxer `turcbtrv.c` que conté el codi de la funció BTRV per a Turbo C.

Trobareu el material necessari per a gestionar fitxers Btrieve en MS-DOS via programes desenvolupats en Turbo C, en el contingut "Material Btrieve per a MS-DOS i Turbo C" de la web d'aquest crèdit.

Anem a definir ara un fitxer Btrieve per a emmagatzemar dades de persones definides pels tipus de dades següents.

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

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

En unitats didàctiques precedents s'ha anat treballant amb aquests tipus de dades i, per tant, disposem d'una colecció de funcions que hem d'aprofitar. Les podeu trobar declarades en els fitxers `generic2.h`, `data1.h` i `persona2.h` i definides en els corresponents `generic2.c`, `data1.c` i `persona2.c`, en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Es faciliten els fitxers codificats en OEM per a la correcta visualització en MS-DOS.

L'analista de les aplicacions a desenvolupar ha decidit que el fitxer de persones ha de tenir tres vies d'accés:

- Pel NIF, ascendentment, sense valors duplicats i sense poder-los modificar.
- Pel cognom1, cognom2 i nom (és a dir, alfabèticament) ascendentment, amb valors duplicats i podent-los modificar.
- Per data de naixement descendent, amb valors duplicats i podent-los modificar.

Per a crear el fitxer Btrieve podem seguir dos camins: utilitzar l'eina `butil -create` o fer un programa en llenguatge C amb el mateix objectiu. Evidentment, utilitzarem l'eina `butil -create` perquè és el camí més ràpid.

Recordem-ne la sintaxi:

```
butil -create <fitxer_Btrieve> <fitxer_descripció>
```

Per tant, hem de tenir el fitxer descripció per a poder crear el fitxer Btrieve. Podem decidir qualsevol nom per al fitxer Btrieve (compte de no

utilitzar l'extensió .pre i el mateix nom que el fitxer de transaccions que es faci servir, en cas de treballar amb transaccions, en la càrrega de l'SGF). És convenient utilitzar sempre noms que facin referència al contingut. Així, anomenarem `PERSONES.BTR` el fitxer Btrieve i `PERSONES.DES` el fitxer descriptor. Recordem que el fitxer descriptor és un fitxer de text. El seu contingut, per a aconseguir les especificacions demanades, ha de ser:

```
record=81 variable=n key=3 page=2048

position=1 length=10 duplicates=n modifiable=n
type=zstring alternate=n null=n segment=n

position=11 length=21 duplicates=y modifiable=y
type=zstring alternate=n null=n segment=y

position=32 length=21 duplicates=y modifiable=y
type=zstring alternate=n null=n segment=y

position=53 length=16 duplicates=y modifiable=y
type=zstring alternate=n null=n segment=n

position=69 length=4 duplicates=y modifiable=y
type=date descending=y alternate=n null=n segment=n
```

Comentem-ne les diferents parts:

- L'entrada `record` té com a valor `81`, perquè és la quantitat de bytes que ocupa una persona. És el resultat de `sizeof (struct t_persona)`.

- L'entrada `variable` té com a valor `n`, ja que no admetem registres de longitud variable.

- L'entrada `key` té com a valor `3`, ja que definirem tres vies d'accés.

- L'entrada `page` té com a valor 2048, ja que el càlcul de grandària òptima és:

  Longitud de registre lògic = 81 bytes
  Longitud de registre físic = 81 + 8 * 2 = 97 bytes
  El sumatori de 8*2 és degut a tenir dues vies d'accés que han d'admetre valors duplicats.

  La taula 15 efectua els càlculs per a les diverses possibilitats de grandària de pàgina per tal d'escollir l'optima.

  L'opció òptima segons el càlcul de la columna D (taula 15) és utilitzar una pàgina de grandària 2048, ja que només perdem 5 bytes per pàgina.

A l'apartat 2.4.3. d'aquesta unitat didàctica hem presentat els càlculs per a obtenir la grandària òptima de pàgina.

Taula 15. Càlculs per a escollir la grandària de pàgina òptima en un fitxer Btrieve

A	B	C	D
Grandària de pàgina.	Espai que queda per pàgina després de reservar 6 bytes de capçalera per gestió del mateix SGF.	Quocient de la divisió entera de la columna B entre la longitud de registre físic (97), és a dir, quantitat de registres que hi haurà a cada pàgina.	Residu de la divisió entera de la columna B entre la longitud de registre físic, és a dir, quantitat de bytes perduts per pàgina.
512	506	5	21
1024	1018	10	48
1536	1530	15	75
2048	2042	21	5
2560	2554	26	32
3072	3066	31	59
3584	3578	36	86
4096	4090	42	16

- Després, ja comencen les entrades corresponents a un bloc d'especificació per cada segment de clau. En primer lloc, la clau formada per un únic segment corresponent al NIF. Com que està en primer lloc serà la clau zero. En segon lloc (clau 1), la clau corresponent a la classificació alfabètica de les persones. És formada per tres segments de clau (cognom1, seguit de cognom2 i de nom –l'ordre és fonamental–). En tercer lloc (clau 2), la via d'accés corresponent a la data de naixement.

- La clau 1 és constituïda per tres segments, cadascun dels quals és considerat del tipus `zstring` (cadena del llenguatge C) consecutius dins el fitxer. Per què considerem tres segments i no posem un únic segment que comenci a la posició 11 i tingui longitud 21 + 21 + 16 = 58 ? Si ho féssim d'aquesta manera, Btrieve, en trobar el '\0' del `cognom1`, menysprearia la resta de contingut dels camps `cognom2` i `nom`.

- La clau 2 correspon al tipus de dada `date`, ja que ens interessa utilitzar la potència de tractament de Btrieve per a les dates. Ara bé, això és possible perquè la definició del tipus de dada `data` que proposem coincideix exactament amb la definició del tipus `date` que sap gestionar l'SGF Btrieve. Quina casualitat, oi?

```
struct t_data
{
   unsigned char dia, mes;
   unsigned int any;
};
```

Ja podem crear el fitxer. Executem la instrucció corresponent suposant que tenim accés a la utilitat `butil.exe`:

```
C:\> butil -create persones.btr persones.des
```

Lògicament ens apareixerà l'error:

```
Btrieve is not loaded.
```

No hem pensat a carregar l'SGF Btrieve. Carreguem-lo!

```
C:\> btrieve
```

Si tot ha anat bé, tornem a executar la utilitat `butil.exe` per a crear el fitxer. Recordem que el fitxer s'ha definit amb 2048 com a grandària de pàgina i l'SGF s'ha carregat per a poder gestionar pàgines de 2048 com a grandària màxima (valor per defecte de càrrega si no s'especifica el paràmetre /P). Per tant, cap problema. Tot va bé i se us ha d'haver creat el fitxer de nom `PERSONES.BTR`. A partir d'ara, el fitxer `PERSONES.DES` no el necessitarem més, tot i que és bo guardar-lo per si cal tornar a crear el fitxer.

Executem l'eina `butil -stat` per a veure l'estat del fitxer `PERSONES.BTR` i obtenim:

```
C:\>butil -stat persones.btr
Btrieve Utilities Version 5.12
Copyright 1982 - 1990, Novell, Inc.  All Rights Reserved.

 File Stats for persones.btr

     Record Length = 81            Compressed Records = No
  Variable Records = No
    Number of Keys = 3
         Page Size = 2048               Unused Pages = 0
     Total Records = 0

   Key  Position  Length  Duplicates  Modifiable   Type    Null      Total

     0      1       10        No          No      Zstring   --            0
     1     11       21        Yes         Yes     Zstring   --            0
     1     32       21        Yes         Yes     Zstring   --            0
     1     53       16        Yes         Yes     Zstring   --            0
     2     69        4        Yes         Yes     Date    < --            0

 < Indicates descending key segment
```

Observeu que ens està donant la informació necessària per a crear el fitxer i la informació sobre el contingut actual del fitxer:

– Ens diu el nombre total de registres actual dins el fitxer en l'entrada `Total Records`.

– Ens diu, per a cada segment de clau, els valors diferents actuals enregistrats en l'índex corresponent. Així, com que la via d'accés 0 no admet duplicats ni valor nul, el nombre de valors per a aquesta clau sempre serà obligatòriament igual al nombre total de registres del fitxer. Les vies d'accés 1 i 2, però, com que tenen la possibilitat d'admetre duplicats, poden tenir un nombre de valors enregistrats menor que el nombre de registres del fitxer.

– Ens diu, també, el nombre total de pàgines no utilitzades. En aquest moment està a zero, ja que quan l'hem creat no li hem comunicat que volíem fer cap reserva d'espai inicial ni tampoc no hem efectuat insercions i posteriors esborraments.

Ara que ja tenim el fitxer creat, ens interessa omplir-lo. Podríem dissenyar un programa per a gestionar el fitxer, efectuant altes, baixes, consultes i modificacions. Això és cosa vostra. Programar en un SGF seqüencial indexat, després d'haver lluitat en el terreny dels seqüencials i dels relatius, on tot ho ha de controlar el programa, és una tasca fàcil. Farem, però, dos programes perquè pugueu comprovar que l'SGF indexat classifica, evidentment, la informació segons els paràmetres requerits.

L'ordre apropiat per a carregar Btrieve i Turbo C++ mentre s'està desenvolupant les aplicacions, és:

1) Obrir una consola MS-DOS
2) Carregar Btrieve amb els paràmetres que correspongui
3) Carregar Turbo C++ per a desenvolupar els programes (editar, compilar, enllaçar i executar).
4) Descarregar Turbo C++
5) Descarregar Btrieve
6) Tancar la consola MS-DOS
És molt important seguir aquest ordre ja que en executar l'aplicació des de Turbo C++ ja tenim Btrieve carregat. No és correcte carregar Btrieve des de dins Turbo C++ mitjançant l'opció User screen (ALT+F5). ‼

Bé, doncs, anem a per feina i desenvolupem els dos programes que havíem comentat.

a) Exemple d'inserció massiva d'informació

Interessa omplir el fitxer amb un conjunt ampli d'informació i sense haver d'anar efectuant les altes, una a una.

Per a aconseguir-ho, amb qualsevol editor de textos crearem un fitxer de nom `cognoms.txt` que contingui una quantitat determinada de

cognoms, un per a cada línia. Anàlogament, crearem un fitxer `noms.txt` que contingui una quantitat determinada de noms, un per a cada línia.

Dissenyem ara un programa que ompli el fitxer `PERSONES.BTR` amb dades. Aquest programa crearà aleatòriament el NIF, la data de naixement, el pes, l'altura i prendrà el nom i els cognoms de manera aleatòria a partir dels fitxers de text abans creats. Fixeu-vos que en aquest programa estarem treballant simultàniament amb dos SGF: el que aporta el llenguatge C i l'SGF Btrieve.

Trobareu el fitxer `u6n2p01.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.

Es faciliten els fitxers codificats en OEM per a la correcta visualització en MS-DOS.

```
/* u6n2p01.c */

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

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

#include "turcbtrv.c"

#define MAXCOGNOM 24 /* Nombre de cognoms del fitxer COGNOMS.TXT */
#define MAXNOM 24 /* Nombre de noms del fitxer NOMS.TXT */

char fper[128];
int lper;
struct t_persona rper;
char nom[]="PERSONES.BTR";

void obrir_fitxer()
{
   int error;
   char pas[]="";
   int lpas=0;

   error=BTRV(0,fper,pas,&lpas,nom,0);
   if (error==20)
   {
      missatge_ae("SGF Btrieve no carregat! ");
      exit(1);
   }
   if (error)
   {
      char s[80];
      sprintf(s,"Error %d en obrir el fitxer %s. ",error,nom);
      missatge_ae(s);
      exit(1);
   }
}

void tancar_fitxer()
{
   int error;
```

```
   error=BTRV(1,fper);
   if (error)
   {
      char s[80];
      sprintf(s,"Error %d en tancar el fitxer %s. ",error,nom);
      missatge_ae(s);
      exit(1);
   }
}

void main()
{
   unsigned int nI,nJ,nK,error,nErr;
   FILE *fnom, *fcognom;
   unsigned int nreg;   /* Nombre de registres a insertar */
   char clau0[10];

   clrscr(); nErr = 0;

   if ((fnom=fopen("noms.txt","rt"))==NULL)
   {
      missatge_ae("Error en obrir el fitxer NOMS.TXT. ");
      exit (1);
   }

   if ((fcognom=fopen("cognoms.txt","rt"))==NULL)
   {
      missatge_ae("Error en obrir el fitxer COGNOMS.TXT. ");
      exit (1);
   }

   /* Efectuem obertura en mode text perque el llenguatge transformi la
   possible parella CR+LF de final de línia dels fitxers textuals en un únic
   caràcter \n */

   randomize();
   obrir_fitxer();
   gotoxy (5,5); printf ("Quants registres vol insertar?");
   do
   { gotoxy (36,5);nI=scanf("%u",&nreg);fflush(stdin);} while (nI==0);
   gotoxy (5,7); printf ("Registres inserits:");
   gotoxy (5,9); printf ("Intents d'inserció erronis per duplicat de nif:");

   for(nI=0;nI<nreg;)
   {
      /* Construcció aleatòria del nif */
      nJ=1+random(8);strcpy(rper.nif,"000000000");
      for(nK=9-nJ;nK<=8;nK++) rper.nif[nK]=48+random(10);

      /* Construcció aleatòria del cognom1 a partir del fitxer de cognoms */
      nJ=random(MAXCOGNOM); rewind(fcognom);
      for (nK=0;nK<nJ; nK++) fgets(rper.cognom1,21,fcognom);
      fgets(rper.cognom1,21,fcognom);
      rper.cognom1[strlen(rper.cognom1)-1]='\0';

      /* Construcció aleatòria del cognom1 a partir del fitxer de cognoms */
      nJ=random(MAXCOGNOM); rewind(fcognom);
      for (nK=0;nK<nJ; nK++) fgets(rper.cognom2,21,fcognom);
      fgets(rper.cognom2,21,fcognom);
```

```
      rper.cognom2[strlen(rper.cognom2)-1]='\0';

      /* Construcció aleatòria del cognom1 a partir del fitxer de cognoms */
      nJ=random(MAXNOM); rewind(fnom);
      for (nK=0;nK<nJ; nK++) fgets(rper.nom,sizeof(rper.nom)-1,fnom);
      fgets(rper.nom,16,fnom);
      rper.nom[strlen(rper.nom)-1]='\0';

      /* Construcció aleatòria de la data de naixement */
      rper.data_naix.any=1900+random(100);
      rper.data_naix.mes=(unsigned)(1+random(12));
      rper.data_naix.dia=(unsigned)(1+random(27));

      /* Construcció aleatòria del pes i de l'alçada */
      rper.pes=random(256);
      rper.alt=random(256);

      lper=sizeof(rper);
      error=BTRV(2,fper,&rper,&lper,clau0,0);
      switch (error)
      {
         char s[80];
         case 0: nI++; gotoxy (28,7); printf ("%d",nI); break;
         case 5: nErr++; gotoxy (55,9); printf ("%d",nErr); break;
         otherwise:
            sprintf(s,"Error %d en insertar al fitxer %s.",error,nom);
            missatge_ae(s);
      }
   }
   tancar_fitxer();
}
```

Les dues constants simbòliques `MAXCOGNOM` i `MAXNOM` estan definides amb les respectives quantitats de cognoms i noms que hi ha en els fitxers textuals corresponents. Si no coincideixen amb la realitat, no passa res. Simplement s'utilitzen perquè el generador de números aleatoris faciliti valors vàlids per a agafar el corresponent cognom o nom del fitxer. Si la constant és més petita que la quantitat real de cognoms/noms, els cognoms/noms situats en una posició posterior al valor de la constant no seran mai escollits. Per contra, si la constant és més gran que la quantitat real de cognoms/noms, és molt possible que s'esculli moltes vegades el darrer cognom/nom, ja que el programa, si intenta accedir més enllà del final, es quedarà amb el contingut de la darrera línia. També es podria calcular el nombre de cognoms i noms existents en els fitxers textuals per no haver de declarar aquestes constants...

Cal fer notar, en primer lloc, que en el programa anterior s'utilitzen funcions del llenguatge C que encara no havien aparegut, però es considera que ja heu de tenir la suficient capacitat per a anar utilitzant les diferents funcions que el llenguatge C aporta en les seves llibreries. ¿Ja sabeu utilitzar l'ajuda que aporta el llenguatge C, oi?

Executeu el programa i demaneu que insereixi una quantitat considerable de registres (uns 200, per exemple). Veureu que per pantalla va informant del nombre de registres inserits i del nombre de registres rebutjats per duplicitat de valor per la clau 0, és a dir, pel camp NIF. Us proposem que modifiqueu el programa perquè, en obtenir aleatòriament el NIF, comprovi si ja existeix aquest NIF, i, si és així, intenti obtenir un altre NIF abans de generar la resta de dades del

registre i de fer l'enregistrament al fitxer. Ara bé, penseu quina modificació fareu perquè pugui funcionar també en un sistema informàtic en xarxa o multiusuari. Hi ha alguna diferència en el disseny?

Una vegada el fitxer tingui dades, comproveu-ne l'estat amb la utilitat `butil -create`. Si el fitxer té un nombre considerable de registres, podreu observar que el nombre de registres coincideix amb el nombre de valors diferents per la clau 0, ja que aquesta clau no admet duplicats ni valor nul. Ara bé, les claus 1 i 2 poden tenir valors inferiors al nombre de registres, la qual cosa implica obligatòriament que hi ha valors duplicats (ja que no hi pot haver valors nuls).

b) Exemple de visualització d'informació segons diverses vies d'accés

Dissenyem ara un altre programa que ens permeti comprovar que, efectivament, l'SGF classifica adequadament. Ja que tenim el fitxer `PERSONES.BTR` amb tres vies d'accés, elaborarem un programa que permeti llistar per pantalla el contingut del fitxer segons els tres criteris de classificació.

Però abans de veure la codificació del programa, penseu en el seu disseny. Cal accedir al primer registre (`llegir_inferior` - operació 12) per la corresponent via d'accés i, posteriorment, fer un recorregut seqüencial (`llegir_següent` - operació 6) per la mateixa via d'accés. Per a poder efectuar aquest tractament, caldrà treballar amb diferents vies d'accés, i ja sabeu que l'argument `àrea_clau` de la funció `BTRV` ha de ser versàtil ja que quan treballem amb la clau 0 (`nif`) ha d'emmagatzemar una cadena de 10 caràcters, quan treballem amb la clau 1 (`cognom1+cognom2+nom`) ha d'emmagatzemar tres cadenes i quan treballem amb la clau 2 (`data_naix`) ha d'emmagatzemar una dada de tipus `t_data`. Com ho hem de fer en llenguatge C?

Cal definir per cada clau d'accés una variable adequada:

```
char clau0[10];
struct t_clau1 { char c1[21], c2[21], n[16]; } clau1;
struct t_data clau2;
```

És a dir, tenim les variables `clau0`, `clau1` i `clau2` per a treballar amb cada via d'accés. És clar, d'altra banda, que el programa que volem efectuar ha de tenir tres opcions (llista per a les tres vies d'accés) i l'execució de cadascuna és anàloga a la de les altres, exceptuant la variable a passar en l'argument `àrea_clau` i, evidentment, el valor a passar en l'argument `núm_clau`. Però això no representa cap problema, perquè podem passar una variable entera que contingui el valor 0, 1 o 2 segons interessi. La qüestió és si podem aconseguir amb una única crida

a la funció BTRV que el paràmetre àrea_clau faci referència a clau0, clau1 o clau2 segons interessi. La resposta és SÍ, amb la utilització d'un void *.

Simplificant, només us cal saber que podeu definir:

```
void *clau;
```

Llavors, en lloc de posar &clau0, &clau1 o &clau2 podem posar simplement clau si abans s'ha tingut la precaució d'omplir aquest punter adequadament amb clau = &clau0 o clau = &clau1 o clau = &clau2.

Vegem el programa en llenguatge C:

```
/* u6n2p02.c */

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

#include "turcbtrv.c"

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

char fcl[128];
int lper;
struct t_persona rper;
char clau0[10];
struct t_clau1 { char c1[21], c2[21], n[16]; } clau1;
struct t_data clau2;
char nom[]="PERSONES.BTR";

void centrar(char *s, int fil, int col, int l)
{
   char cadena[80];
   int k;
   if (l>79) l=79;
   k=strlen(s);
   cadena[0]='\0';
   while (strlen(cadena)<(l-k)/2) strcat(cadena," ");
   strcat(cadena,s);
   while (strlen(cadena)<l) strcat(cadena," ");
   gotoxy(col,fil);cputs(cadena);
}

void titol(char *s)
{
   window(1,1,80,4);textattr(27);clrscr();
   centrar(s,1,1,79);
}
```

Trobareu el fitxer u6n2p02.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.

Es faciliten els fitxers codificats en OEM per a la correcta visualització en MS-DOS.

```
char pantalla_menu()
{
   char cOpcio;
   window(1,1,80,25);textattr(15);clrscr();
   gotoxy( 19, 5 );
   cputs( "+-----------------------------------------+" );
   gotoxy( 19, 6 );
   cputs( "¦                                         ¦" );
   gotoxy( 19, 7 );
   cputs( "¦     LLISTATS DE PERSONES PER PANTALLA    ¦" );
   gotoxy( 19, 8 );
   cputs( "+-----------------------------------------¦" );
   gotoxy( 19, 9 );
   cputs( "¦                                         ¦" );
   gotoxy( 19, 10 );
   cputs( "¦  1. Classificats per NIF                ¦" );
   gotoxy( 19, 11 );
   cputs( "¦  2. Classificats per COGNOMS-NOM        ¦" );
   gotoxy( 19, 12 );
   cputs( "¦  3. Classificats per DATA DE NAIXEMENT  ¦" );
   gotoxy( 19, 13 );
   cputs( "¦  0. Sortir                              ¦" );
   gotoxy( 19, 14 );
   cputs( "¦                                         ¦" );
   gotoxy( 19, 15 );
   cputs( "¦     Premi Opció:                        ¦" );
   gotoxy( 19, 16 );
   cputs( "+-----------------------------------------+" );
   do
   {
   gotoxy( 38, 15 );cOpcio=getche();
   } while (cOpcio<'0' || cOpcio>'3');
   return cOpcio;
}

void obrir_fitxer()
{
   int error;

   error=BTRV(0,fcl,&rper,&lper,nom,0);
   if (error==20)
   {
      missatge_ae("SGF Btrieve no carregat! ");
      exit(1);
   }
   if (error)
   {
      char s[80];
      sprintf(s,"Error %d en obrir l'arxiu %s. ",error,nom);
      missatge_ae(s);
      exit(1);
   }
}

void tancar_fitxer()
{
   int error;
   error=BTRV(1,fcl);
   if (error)
```

```
   {
      char s[80];
      sprintf(s,"Error %d en tancar l'arxiu %s. ",error,nom);
      missatge_ae(s);
      exit(1);
   }
}

void main()
{
   int error,lin;
   char cOpcio;
   clrscr();
   obrir_fitxer();
   lper=sizeof(rper);
   while ((cOpcio=pantalla_menu())!='0')
   {
      void *clau;
      char s[80];
      int nClau;
      clrscr();
      switch (cOpcio)
      {
         case '1': nClau=0; clau=clau0;
         strcpy(s,"LLISTAT PERSONES CLASSIFICADES PER NIF - ASCENDENTMENT");
            break;
         case '2': nClau=1; clau=&clau1;
         strcpy(s,"LLISTAT PERSONES CLASSIFICADES PER COGNOMS-NOM - ASCENDENTMENT");
            break;
         case '3': nClau=2; clau=&clau2;
         strcpy(s,"LLISTAT PERSONES CLASSIFICATS PER DATA NAIXEMENT - DESCENDENTMENT");
         break;
      }
      titol(s);
      gotoxy(1,3); cputs("NIF       COGNOM 1             COGNOM 2             NOM
D.NAIXEMENT  ");
      gotoxy(1,4); cputs("-------------------------------------------------------------------------------
-----");
      lin=1; window(1,5,80,24); textattr(15); clrscr();
      error=BTRV(12,fcl,&rper,&lper,clau,nClau);
      while (!error)
      {
         if (lin==21)
         {
            window(1,1,80,25); missatge_ae("");
            lin=1; window(1,5,80,24); textattr(15); clrscr();
         }
         outputPersonaTab(lin,rper);
         error=BTRV(6,fcl,&rper,&lper,clau,nClau);
         lin++;
      }
      window(1,1,80,25);
      if (error!=9)
      {
         char s[80];
         sprintf(s,"Error %d en consultar el fitxer %s  ",error,nom);
         missatge_ae(s);
      }
      else
```

```
      {
         char s[80];
         sprintf(s,"Final dels registres al fitxer %s   ",nom);
         missatge_ae(s);
      }
   }
   tancar_fitxer();
   clrscr();
}
```

Comentar que ens hem pres la llicència d'utilitzar funcions específiques de Turbo C++ no existents en gcc, donat que l'accés a Btrieve només el podem practicar en la plataforma Turbo C++.