Tema 9: Fitxers

Tema 9: Fitxers

1. Treballar amb fitxers

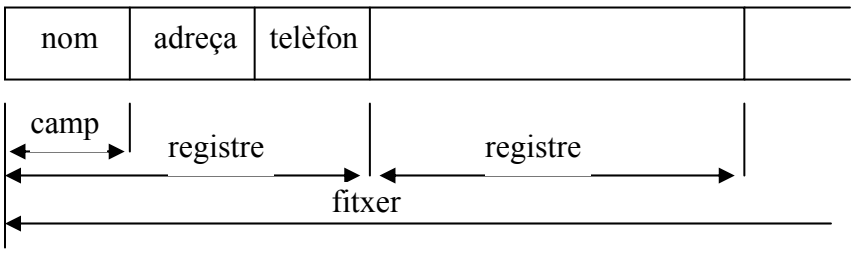
Tots els programes realitzats fins ara obtenien les dades necessàries per a la seva execució de l'entrada estàndard (el teclat) i visualitzaven els resultats en la sortida estàndard (la pantalla). Per una altra banda, una aplicació podrà retenir les dades que manipula en el seu espai de memòria, només mentre estigui en execució; és a dir, qualsevol dada introduïda es perdrà quan l'aplicació finalitzi.

Per exemple, si hem realitzat un programa amb la intenció de construir una agenda, l'executem i emmagatzemem les dades *nom*, *adreça* i *telèfon* de cadascun dels components de l'agenda en una matriu, les dades estaran disponibles mentre el programa estigui en execució. Si finalitzem l'execució del programa i l'executem de nou, haurem de tornar a introduir de nou totes les dades.

La solució per a fer que les dades persisteixin d'una execució per a una altra és emmagatzemar-los en un fitxer en el disc en vegada de en memòria. Aleshores, cada vegada que s'executi l'aplicació que treballa amb aquestes dades, podrà llegir del fitxer les que necessiti i manipular-les. Nosaltres procedim de forma anàloga en molts aspectes de la vida ordinària; emmagatzemem les dades en fitxes i guardem el conjunt de fitxes en allò que generalment anomenem fitxer o arxiu.

Des del punt de vista informàtic, un fitxer o arxiu és una col·lecció d'informació que s'emmagatzema en un suport, generalment magnètic, per a poder-la manipular en qualsevol moment. Aquesta informació s'emmagatzema com un conjunt de registres, contenint tots ells, generalment, els mateixos camps. Cada camp emmagatzema una dada d'un tipus de dades. El registre més simple estaria format per un caràcter.

Per exemple, si volguéssim emmagatzemar en un fitxer les dades relatives a l'agenda de telèfons a què ens hem referit anteriorment, podríem dissenyar cada registre amb els camps *nom*, *adreça* i *telèfon*. Segons això i des d'un punt de vista gràfic, pot imaginar-se l'estructura del fitxer així:



Cada camp emmagatzemarà la dada corresponent. El conjunt de camps descrits forma allò que hem anomenat registre, i el conjunt de tots els registres formen un fitxer que emmagatzemarem, per exemple, en el disc sota un nom.

Per tant, per a manipular un fitxer que identifiquem amb un nom, són tres les operacions que hem de realitzar: obrir el fitxer, escriure o llegir registres del fitxer i tancar el fitxer. En la vida ordinària fem el mateix, obrim el calaix que conté les fitxes (fitxer), agafem una fitxa (registre) per a llegir dades o escriure dades i, finalitzat el treball amb la fitxa, la deixem al seu lloc, continuant amb altres fitxes; una vegada acabat el treball, tanquem el calaix de fitxes (fitxer).

2. Els arguments de la funció `main()`

És útil, moltes vegades, especificar opcions o valors en el moment d'executar un programa des de la línia de comandaments del sistema operatiu. Els comandaments de MSDOS com `copy` o `rename` poden ser exemples d'execució d'un "programa" amb arguments entrats junts al nom del programa. El programa reconeixerà aquests paràmetres a través dels arguments de la funció `main()`.

La forma de declarar els arguments de la funció `main()` és:

```
void main(int argc, char *argv[]);
```

El primer paràmetre `argc` és el nombre d'arguments que s'han especificat a la línia de comandaments (el nom del programa executable compta com el primer argument). El segon paràmetre `argv` és un vector de cadenes de caràcters que conté els arguments especificats. Exemple, si a la línia de comandaments s'escriu:

```
programa arg1 arg2 arg3
```

El primer paràmetre, `argc` serà 4, i el segon paràmetre, `argv[]` contindrà la llista `programa arg1`, `arg2`, `arg3`. És a dir, `argv[0]="programa"`, `argv[1]="arg1"`, `argv[2]="arg2"`, `argv[3]="arg3"`.

Si aquests paràmetres són nombres, s'han de convertir al tipus corresponent amb les funcions de les llibreria estàndard `math.h` i `stdlib.h`:

`float atof(char*)`	Converteix una cadena en un nombre del tipus `float`.
`int atoi(char*)`	Converteix una cadena en un nombre del tipus `int`.
`double strtod(char*)`	Converteix una cadena en un nombre del tipus `double`.

Exemple:

```
/* Arguments de la funció main() */

#include <stdio.h>

void main (int argc, char *argv[]){
   int i;

   printf("El nombre total d'arguments és %d\n", argc);

   for(i=0; i<argc; i++)
      printf("L'argument %d és %s\n", i, argv[i]);
}
```

3. Treball bàsic amb fitxers

3.1. Fluxos d'entrada / sortida

El sistema d'entrada i sortida de C proporciona un sistema independent del dispositiu físic en el qual, o des del qual, es realitzin les operacions. Aquest sistema independent és una abstracció de

dispositiu que rep el nom de flux o corrent (en anglès `stream`). De fet, només necessitem considerar un arxiu com un flux de dades que circulen en un o altre sentit.

Tècnicament, quan s'obre un arxiu per a utilitzar funcions d'entrada i sortida, aquest arxiu s'associa amb una estructura del tipus `FILE` (definit a `stdio.h`). Aquesta estructura conté tota la informació bàsica sobre l'arxiu.

Quan un programa C comença la seva execució, s'obren automàticament alguns fluxos, vinculats amb diversos dispositius. Aquestos fluxos i els dispositius vinculats per defecte són els següents:

Flux	Dispositiu al que està vinculat
`stdin`	dispositiu d'entrada estàndard (teclat)
`stdout`	dispositiu de sortida estàndard (pantalla)
`stderr`	dispositiu d'error estàndard (pantalla)
`stdprn`	dispositiu d'impressió estàndard (port paral·lel)

3.2. Declaració d'un arxiu

Com tota variable, una variable arxiu ha de ser declarada abans d'utilitzar-la. Així, la sentència

```
FILE *f;
```

declara una variable d'arxiu anomenada `f`. Aquesta variable, de fet, és un punter a l'objecte `FILE`, que és una estructura definida a l'arxiu capçalera `stdio.h`. Per tant, és necessari incloure la directiva `#include <stdio.h>` en qualsevol programa en què s'utilitzin arxius.

3.3. Obertura d'un arxiu

Declarar una variable d'arxiu no és suficient per poder treballar amb ell. És necessari precisar dues coses:

- D'una banda, és imprescindible assenyalar de quin arxiu físic es tracta. És necessari donar un nom a l'arxiu que serà el que restarà al suport magnètic.
- D'altra banda, farà falta precisar el tipus de treball que volem fer: llegir, escriure,...

Aquestes característiques es donen amb l'ajuda de la funció `fopen()` que té la següent sintaxi:

```
FILE *fopen(char *nom_arxiu, char *mode)
```

La funció `fopen()` retornarà el valor `NULL` si no s'ha pogut obrir l'arxiu, per exemple, si el disc està protegit o ple.

El primer argument de la funció `fopen()` és el nom físic de l'arxiu i permet l'associació entre el nom físic sobre el suport (`nom_arxiu`) i el nom lògic de l'arxiu en el programa.

El segon argument està format per una cadena de caràcters que precisa el mode de treball amb l'arxiu. Són possibles les diferents possibilitats:

`r` (*read*). Obre un arxiu només per llegir. L'arxiu ha d'existir. El cursor se situarà damunt del primer byte de l'arxiu.

`w` (*write*). Obre un arxiu només per escriure. El cursor se situarà damunt del primer byte de l'arxiu. Si l'arxiu ja existeix es sobreescriu.

`a`	(*append*). Obre un arxiu per escriure. El cursor se situarà damunt de l'últim byte de l'arxiu si aquest ja existeix, per tant, serveix per afegir dades a un arxiu. En el cas que l'arxiu no existeixi funciona igual que `w`.
`r+`	Obre un arxiu per llegir i escriure alhora. L'arxiu ha d'existir prèviament. El cursor se situa al començament de l'arxiu.
`w+`	Obre un arxiu per llegir i escriure alhora. Si l'arxiu existeix prèviament, s'esborra el seu contingut.
`a+`	Obre un arxiu per afegir dades i per llegir. Si l'arxiu no existeix el crearà.

A les formes d'accés mencionades, se les pot afegir un caràcter `t` o `b` (per exemple, `rb`, `a+b` o `ab+`), per a indicar si el fitxer s'obre en mode *text* o en mode *binari*. Si `t` o `b` no s'especifiquen, per defecte s'agafa el mode *text*.

Si s'obre un fitxer per a actualització (+), entre una operació de lectura i una d'escriptura, o viceversa, s'ha de cridar a la funció `fseek()` o la a funció `rewind()`.

Exemple:

```
FILE *f;
if((f=fopen("fitxer.dat","rb")==NULL)
{
   printf("S'ha produït un error en obrir el fitxer");
   return;
}
```

3.4. Fitxers en mode *text* i en mode *binari*

Un fitxer de text emmagatzema tota la informació en format caràcter. Per exemple, els valors numèrics es converteixen en els caràcters (dígits) que són la representació numèrica. S'indica el mode text en el segon argument de `fopen()`, amb una `t`.

Un fitxer binari emmagatzema tota la informació utilitzant la mateixa representació binària que la computadora utilitza internament. Els fitxers binaris són més eficients, no hi ha conversió entre la representació en la computadora i la representació en el fitxer; també ocupen menys espai. S'indica el mode binari en el segon argument de `fopen()`, amb una `b`.

Treballarem en mode *text* quan vulguem poder accedir al fitxer des d'una aplicació externa (per exemple el Bloc de notes o l'Access). En cas contrari, sempre convindrà treballar en mode *binari*.

Quan estem treballant amb un compilador C sota el sistema operatiu Windows (o MS-DOS) s'han de tenir en compte les següents consideracions:

A diferència de UNIX, en Windows un fitxer pot ser obert en mode *text* o en mode *binari*. La necessitat de dos modes diferents, és per les incompatibilitats existents entre C i Windows ja que C fou dissenyat originalment per al sistema operatiu UNIX. En el mode *text*, quan s'envia a la sortida el caràcter `'\n'`, utilitzat en C per a canviar de línia, és traduït en dos caràcters (*CR+LF*) i a la inversa, la combinació *CR+LF* és traduïda en un únic caràcter `'\n'` (*LF*) quan es tracta d'una entrada de dades. Això significa que en Windows, quan un programa C escriu en un fitxer tradueix el caràcter `'\n'` en els caràcters *CR+LF*; i quan C llegeix des d'un fitxer i troba els caràcters *CR+LF*, els tradueix a `'\n'`. Aquesta traducció pot ocasionar problemes quan aquests caràcters no

es corresponen amb text, sinó que, per exemple, formen part d'un valor numèric que estem recuperant des d'un fitxer en disc, i dos dels bytes que componen aquest valor coincideixin amb la representació del caràcters *CR+LF*. Per a evitar aquest tipus de problemes, utilitzeu el mode *binari*, ja que en aquest mode les traduccions indicades no tenen lloc.

3.5. Tancar un arxiu

Al final del treball sobre l'arxiu, o si més no abans de sortir del programa, és necessari tancar l'arxiu. Aquest tancament té diferents objectius. El primer consisteix en alliberar el canal atribuït a l'arxiu i permetre obrir un altre arxiu sense superar el màxim nombre possible d'arxius oberts al mateix temps. El segon, i gairebé sempre el més important, és buidar el buffer (zona de memòria) associat a l'arxiu. L'escriptura d'un arxiu no es fa directament al suport físic sinó que les diferents escriptures són enviades a un buffer per tal d'estalviar accessos al disc que són massa lents. En cas d'acabament normal del programa, el sistema operatiu s'encarrega de tancar els arxius, però això no sempre és possible, per exemple quan marxa el llum, si es bloqueja l'ordinador, etc. Per tant és recomanable cada cert interval de temps tancar i tornar-lo a obrir per evitar pèrdua de dades. A més, un fitxer obert per una aplicació, generalment no està disponible per a una altra fins que no es tanqui.

Per tancar un fitxer, ho farem amb la funció `fclose()` que té la següent sintaxi:

```
int fclose (FILE *nom_arxiu);
```

Aquesta funció torna el valor enter 0 si l'arxiu s'ha pogut tancar correctament.

4. Lectura i escriptura de fitxers

4.1. Lectura i escriptura. Fitxers de text i fitxers binaris

Les funcions de lectura i escriptura que s'utilitzen per als fitxers de text són: `fgetc()`, `fputc()`, `fgets()`, `fputs()`, `fscanf()` i `fprintf()`.

Les funcions de lectura i escriptura que s'utilitzen per als fitxers binaris són: `fgetc()`, `fputc()`, `fread()` i `fwrite()`.

4.2. Lectura i escriptura de caràcters: `fgetc()` i `fputc()`

Les funcions `fgetc()` i `fputc()` s'utilitzen per llegir i escriure caràcters en un arxiu obert prèviament amb la funció fopen(). La sintaxi d'aquestes dues funcions són:

```
int fputc(int c, FILE *f)

int fgetc(FILE *f)
```

Si tot funciona correctament, les dues funcions retornen el caràcter escrit o llegit; si es produeix un error o acaba el fitxer retornen `EOF`, una constant entera definida a l'arxiu de capçalera `stdio.h` i que significa el final de l'arxiu.

`EOF` és una constant entera, per a així distingir-la de qualsevol caràcter vàlid llegit, per tant s'ha d'emmagatzemar el valor llegit per aquestes funcions en una variable entera, tot i que després la tractem com si fos caràcter. Si s'utilitzés una variable caràcter, es podria confondre `EOF` amb un altre caràcter, creient haver arribat al final del fitxer abans d'hora.

Exemple:

```
/* Mostra el contingut d'un fitxer per pantalla */
int c;
while ( (c=fgetc(f)) != EOF )
   printf("%c", c);
```

4.3. Lectura i escriptura de cadenes de caràcters: `fgets()` i `fputs()`

Les funcions `fgets()` i `fputs()` permeten llegir i escriure cadenes de caràcters en els arxius prèviament oberts amb la funció `fopen()`. La sintaxi d'aquestes funcions és:

```
char *fgets( char *cadena, int n, FILE *f);
```

```
int fputs( const char *cadena, FILE *f );
```

La funció `fgets()` llegeix de l'arxiu apuntat per `*f` una cadena de caràcters i la guarda a la variable cadena. Si la mida de la cadena llegida supera els *n-1* caràcters, només es llegeixen els *n-1* primers caràcters. Al resultat guardat en `cadena` se li afegeix el caràcter de fi de cadena (`'\0'`). Si s'ha llegit el caràcter de nova línia també és emmagatzemat a la variable.

La funció `fgets()` retorna un punter a la cadena `cadena` o bé un punter `NULL` si hi ha hagut algun error.

La funció `fputs()` escriu el contingut de la variable cadena a l'arxiu apuntat per `*f`. Aquesta funció torna la constant entera `EOF` si troba algun error.

Les funcions `gets()` i `puts()` són semblants a `fgets()` i `fputs()`, però operen sobre `stdin` i `stdout`. De mode desconcertant, `gets()` elimina el `'\n'` terminal i `puts()` l'agrega.

Exemple 1:

```
/* Mostrem un fitxer per pantalla */
while(fgets(frase,81,f)!=NULL)
   puts(frase);
```

Exemple 2:

```
/* Llegim cadenes del teclat i les escrivim en un fitxer */
/* gets() no llegeix el '\n' */
while (gets(cadena)!=NULL)
{
   fputs(cadena,f);
   fputc('\n',f);
}
```

Exemple 3:

```
/* Llegim cadenes del teclat i les escrivim en un fitxer */
/* fgets() sí llegeix el '\n' */
/* Amb fgets() indiquem quants caràcters com a màxim llegim */
while (fgets(cadena,81,stdin)!=NULL)
   fputs(cadena,f);
```

4.4. Lectura i escriptura formatada: `fscanf()` i `fprintf()`

Les funcions `fprintf()` i `fscanf()` funcionen exactament igual que `printf()` i `scanf()` excepte pel fet que necessiten un argument addicional per apuntar a l'arxiu corresponent. La sintaxi d'aquestes funcions és:

```
int fprintf(punter_arxiu,"cadena_de_control",llista arguments);

int fscanf (punter_arxiu,"cadena_de_control",llista arguments);
```

La funció `fprintf()` retorna el nombre de caràcters enviats a l'arxiu. Si es produeix un error retorna un nombre negatiu.

La funció `fscanf()` retorna el nombre d'elements que ha pogut llegir. Quan arriba al final de l'arxiu retorna la constant `EOF`.

Exemple:

```
/* Les dues línies següents són equivalents */
fprintf(stdout, "Num: %d", n);
printf("Num: %d", n);

/* Les dues línies següents són equivalents */
fscanf(stdin, "%d:%d:%d", &h, &m, &s);
scanf("%d:%d:%d", &h, &m, &s);
```

4.5. Lectura i escriptura de registres: `fread()` i `fwrite()`

També hi ha la possibilitat d'escriure o llegir tot un bloc de dades en un arxiu binari. Caràcters, enters, vectors, estructures, vectors d'estructures i altres dades poden ser gravades de cop amb la funció `fwrite()`, o llegides de cop amb la funció `fread()`. De fet, aquestes funcions poden substituir a les funcions d'entrada / sortida estudiades fins ara.

El prototipus de la funció `fwrite()` és:

```
int fwrite(void *p, int mida_element , int nombre_elements, FILE *f)
```

Aquesta funció retorna el nombre d'elements escrits. Si aquesta dada no és l'esperada tenim algun error d'escriptura.

Farem una breu descripció de cada argument:

- Primer argument. `void *p`. És un punter al bloc de dades que volem escriure a l'arxiu. Si l'element a gravar és una estructura hem d'enviar l'adreça de l'estructura.
- Segon argument. `int mida`. Aquesta variable és la mida en bytes de l'element a gravar. Per assegurar la portabilitat és convenient utilitzar l'operador `sizeof()`.
- Tercer argument. `int nombre_elements`. Aquest és el nombre d'elements a gravar. El nombre total de bytes serà el producte d'aquesta variable i l'anterior.
- Quart argument. `FILE *f`. És el punter d'arxiu corresponent.

De la mateixa forma, per llegir tot un bloc es fa servir la funció `fread()`, el prototipus de la qual és:

```
int fread(void *p, int mida_element , int nombre_elements, FILE *f)
```

Aquesta funció retorna el total d'elements llegits. Si aquest nombre és inferior a `nombre_elements` és que s'ha produït un error (aquest error pot ser simplement que ha trobat el final de l'arxiu).

El significat dels arguments és similar als de la funció `fwrite()`, només puntualitzar que `void *p` és ara un punter al lloc de la memòria que s'omplirà amb les dades llegides.

Exemple:

```
fread(&c,sizeof(char),1,f); // llegeix un caràcter

fwrite(v,sizeof(int),10,f); // escriu un vector de 10 enters

fread(&alum,sizeof(struct alumne),1,f); // llegeix una estructura

while (fread(&alum,sizeof(struct alumne),1,f) > 0)
   ...  // mostrem per pantalla les dades
```

5. Altres funcions orientades a fitxers

5.1. Marcador del fitxer: `fseek()`, `ftell()` i `rewind()`

Quan s'obre un arxiu amb la funció `fopen()` es crea un marcador o punter que indica la posició exacta en el flux de dades que es llegirà o escriurà en la pròxima funció d'entrada o sortida. La funció `fopen()` fa que aquest marcador comenci apuntant al principi del flux o al final, segons el mode establert. Les funcions abans tractades, modifiquen automàticament aquest marcador per tal que la següent funció d'entrada o sortida llegeixi o escrigui a continuació.

No obstant això, C incorpora funcions que permeten alterar aquest marcador.

La funció `fseek()` permet modificar la posició d'aquest marcador. La seva sintaxi és:

```
int fseek( FILE *f, long desp, int origen );
```

El punter `f` és un punter del tipus `FILE` que es refereix a un arxiu obert. La variable `desp` és el nombre de bytes que es desplaçarà el marcador des de la posició indicada per la variable origen. La variable `origen` només pot tenir tres valors:

- `SEEK_SET`: es calcula el desplaçament des del principi de l'arxiu.
- `SEEK_CUR`: es calcula el desplaçament des de la posició actual del marcador.
- `SEEK_END`: es calcula el desplaçament des de la posició final de l'arxiu.

La funció `fseek()` torna el valor 0 si el desplaçament s'ha produït amb èxit o un valor diferent de 0 si no s'ha pogut produir.

La funció `fseek()` només treballa correctament en mode *binari*.

Si esteu actualitzant un fitxer, s'ha de cridar a la funció `fseek()` (o `rewind()`) entre una lectura i una escriptura, i entre una escriptura i una lectura.

La funció `ftell()` torna la posició actual del marcador comptant des de la posició inicial. La seva sintaxi és:

```
long ftell( FILE *f);
```

La funció `rewind()` mou el marcador al començament de l'arxiu. La seva sintaxis és:

```
void rewind( FILE *f );
```

Una crida a aquesta funció equival a una crida a `fseek()` amb desplaçament 0 i origen `SEEK_SET`, amb l'excepció de què `rewind()` posa a zero els bits d'error i el bit de fi de fitxer, i `fseek()` no.

Aquestes funcions es poden fer servir per crear arxius d'accés directe, com veurem més endavant.

5.2. Final de fitxer: `feof()`

Quan es crea un fitxer el sistema afegeix automàticament al final del mateix una marca de fi de fitxer. Quan en una operació de lectura sobre un fitxer s'intenta accedir més enllà de la marca de fi de fitxer, automàticament el sistema posa a 1 el bit de *fi de fitxer* de l'estructura `FILE`. Un programa pot conèixer l'estat d'aquest indicador invocant a la funció `feof()`:

```
int feof (FILE *f);
```

Aquesta funció torna un valor diferent de zero (cert) quan s'intenta llegir més enllà de la marca `EOF`. En un altre cas torna 0 (fals).

5.3. Detecció d'errors: `ferror()`

La funció `ferror()` determina si s'ha produït un error en una operació sobre un arxiu. El prototipus de la funció és:

```
int ferror( FILE *f);
```

on `f` és un punter a un arxiu. La funció `ferror()` retorna un valor diferent de zero (cert) si s'ha produït un error en l'última operació sobre l'arxiu, en cas contrari retorna 0 (fals). Per tant, quan vulguem implementar perfectament un programa, en cada operació sobre un arxiu és convenient cridar a la funció `ferror()`.

5.4. Esborrar fitxers: `remove()`

La funció `remove()` té per protocol:

```
int remove(char *nom_arxiu);
```

Aquesta funció esborra l'arxiu que especifiquem en el seu argument. Si tot funciona correctament retornarà zero, en cas contrari un nombre diferent de zero.

5.5. Canvi de nom de fitxers: `rename()`

La funció `rename()` té per protocol:

```
int rename(char *nom_arxiu_nou, char *nom_arxiu_vell);
```

Aquesta funció canvia el nom de l'arxiu de `nom_arxiu_vell` a `nom_arxiu_nou`. Si tot funciona correctament retornarà zero, en cas contrari un nombre diferent de zero.

5.6. Eliminació de registres

No hi ha cap funció específica per a esborrar un registre d'un fitxer. Per a esborrar un registre tenim dues possibilitats:

a) Amb un camp lògic del registre es marca si aquest és vàlid. Per a esborrar un registre posarem el camp a fals. Quan treballem amb els registres del fitxer, no farem cas d'aquells que estiguin marcats com a fals.

b) Es copia tot el fitxer a un fitxer auxiliar, excepte el registre a esborrar. Després s'esborra l'original (funció `remove()`) i es canvia de nom l'auxiliar per l'original (funció `rename()`).

En la pràctica, se sol utilitzar una mescla de les dues opcions. Quan s'esborra un registre s'escull la primera opció, i quan el programa finalitza o quan l'usuari ho desitja, es compacta el fitxer, és a dir, s'esborren físicament els registres marcats, utilitzant la segona opció.

6. Accés seqüencial

El tipus d'accés més simple a un fitxer de dades és el seqüencial: els registres que s'escriuen al fitxer són col·locats automàticament un a continuació d'un altre, i quan es llegeixen, es comença pel primer, es continua amb el següent, i així successivament fins arribar al final. Aquesta forma de procedir possibilita que els registres puguin ser de qualsevol longitud, fins i tot d'un sol byte.

Veiem un exemple que llegeix de l'entrada estàndard grups de dades (registres) definits de la forma que s'indica a continuació i els emmagatzema en un fitxer:

```
struct registre
{
   char nom[40];
   char adreca[40];
   long telefon;
}
```

Per a realitzar aquest exemple, escriurem un programa *escriu.c* amb dues funcions: `creaFitxer()` i `main()`.

La funció `main()`:

- Obté el nom del fitxer des de l'entrada estàndard.
- Invoca a la funció `creaFitxer()` passant com a argument el nom del fitxer.

La funció `creaFitxer()` rep com a paràmetre el nom del fitxer que es desitja crear i realitza les tasques següents:

- Crea un flux cap al fitxer especificat. Si no pot, llança un missatge i surt de la funció.
- Llegeix grups de dades *nom*, *adreça* i *telèfon* de l'entrada estàndard i els escriu en un fitxer.

Veiem el codi de la funció `creaFitxer()`:

```
void creaFitxer(char *nomFitxer)
{
  FILE *f;   // identificador del fitxer
  struct registre reg;      // definir un registre
  int sortir=0;

  // Obrir el fitxer nomFitxer per a escriure "wb"
  if ((f = fopen(nomFitxer, "wb")) == NULL)
  {
    printf("El fitxer no pot obrir-se\n");
    return;
  }

  // Llegir dades de l'entrada estàndard i escriure-les
  // en el fitxer
```

```
  do
  {
    printf("Nom:     "); gets(reg.nom);
    printf("Adreça:  "); gets(reg.adreca);
    printf("Telèfon: "); scanf("%ld", &reg.telefon);
    fflush(stdin);

    // Emmagatzemar un registre en el fitxer
    fwrite(&reg, sizeof(reg), 1, f);

    printf("Desitgeu escriure un altre registre? (s/n) ");
    if(getchar()!='s')
       sortir=1;
    fflush(stdin);
  }
  while (!sortir);
  fclose(f);
}
```

Per a llegir el fitxer creat pel programa anterior, anem a escriure un altre anomenat *llegeix.c*, composat per una funció `main()` anàloga i per una nova funció `mostraFitxer()`.

La funció `mostraFitxer()` rep com a paràmetre el nom del fitxer que es desitja llegir i realitza les tasques següents:

- Crea un flux que permet llegir dades des del fitxer passat com a argument. Si no pot fer-ho, llança un missatge i surt de la funció.
- Llegeix un grup de dades *nom*, *adreça* i *telèfon* des del fitxer i els mostra, repetint aquesta operació fins arribar al final del fitxer.

Veiem el codi de la funció `mostraFitxer()`:

```
void mostraFitxer(char *nomFitxer)
{
  FILE *f; // identificador del fitxer d'entrada
  struct registre reg;     // definir un registre

  // Obrir el fitxer nomFitxer per llegir "rb"
  if ((f = fopen(nomFitxer, "rb")) == NULL)
  {
    printf("El fitxer no pot obrir-se.");
    return;
  }

  // Llegir dades del fitxer i mostrar-les en la sortida estàndard
  while (fread(&reg, sizeof(reg), 1, f) > 0)
  {
    printf("Nom:     %s\n", reg.nom);
    printf("Adreça:  %s\n", reg.adreca);
    printf("Telèfon: %ld\n\n", reg.telefon);
  }

  fclose(f);
```

```
}
```

7. Accés aleatori o directe

Fins aquest punt, en l'accés seqüencial, hem treballat amb el següent esquema: obrir el fitxer, llegir o escriure fins al seu final, i tancar el fitxer. Però no hem llegit o escrit a partir d'una determinada posició dins del fitxer. Això és particularment important quan necessitem modificar alguns dels valors continguts en el fitxer o quan necessitem extraure una part concreta dins del fitxer.

Un fitxer accedit aleatòriament és comparable a un vector. En un vector per a accedir a un dels seus elements utilitzem un índex. En un fitxer accedit aleatòriament l'índex se substitueix per un punter de lectura / escriptura. Aquest punter se situa automàticament al principi del fitxer quan aquest s'obre per llegir i/o escriure, i també es pot moure a qualsevol altra posició en el fitxer invocant a la funció `fseek()`. Una operació de lectura / escriptura comença en la posició on estigui el punter de lectura / escriptura i quan aquesta operació finalitza, la seva posició coincidirà amb el byte just a continuació de l'últim llegit o escrit.

Com a exemple, escriurem un programa *modifica.c* que permeti modificar la llista de telèfons creada amb el programa *escriu.c* realitzat anteriorment. Aquest nou programa obre un fitxer que conté registres de tipus `struct registre` (creat pel programa *escriu.c*), calcula el seu nombre de registres, sol·licita el número de registre que es desitja modificar, el presenta en pantalla i pregunta si se desitja modificar; en cas afirmatiu, sol·licita del teclat les noves dades per a aquest registre i l'emmagatzema en la mateixa posició dins del fitxer reescrivint el registre primitiu. A continuació, torna a preguntar pel número del següent registre a modificar; un valor 0, finalitza el programa. El programa complet es mostra a continuació:

```
/**************** Accés aleatori a un fitxer ****************/
/* modifica.c */

#include <stdio.h>

struct registre
{
  char nom[40];
  char adreca[40];
  long telefon;
};

void mostraReg(FILE *f, int nreg);
void modificaReg(FILE *f, int nreg);

void main()
{
  FILE *f = NULL;              // punter a un flux
  int totalreg = 0;            // nombre total de registres
  int nreg = 0;                // número de registre
  char nomFitxer[30];          // nom del fitxer
  char resposta;

  // Sol·licitar el nom del fitxer
  printf("Nom del fitxer: ");
```

```
  gets(nomFitxer);

  // Obrir el fitxer per llegir i escriure "r+"
  if ((f = fopen(nomFitxer, "r+b")) == NULL)
  {
    printf("Error: no es pot obrir el fitxer\n");
    return;
  }

  // Calcular el nombre total de registres del fitxer
  fseek(f, 0, SEEK_END);
  totalreg = ftell(f)/sizeof(struct registre);

  // Presentar un registre en pantalla i modificar-lo si procedeix
  do
  {
    printf("Num. registre entre 1 i %d (0 per a sortir):",totalreg);
    scanf("%d", &nreg);
    fflush(stdin);
    if ((nreg >= 1) && (nreg <= totalreg))
    {
      mostraReg(f, nreg);
      // Preguntar si es desitja modificar el registre seleccionat
      printf ("Desitgeu modificar aquest registre? (s/n)  ");
      resposta = getchar();
      fflush(stdin);

      if (tolower(resposta) == 's')
         modificaReg(f, nreg);
    }
  }
  while (nreg!=0);
  fclose(f);
}

void mostraReg(FILE *f, int nreg)
{
  int desp; // desplaçament en bytes
  struct registre reg;  // variable de tipus registre
  int bytesPerReg = sizeof(struct registre);

  // Visualitzar un registre
  // Calcula el nombre de bytes fins a la posició indicada
  desp = (nreg - 1) * bytesPerReg;
  // Posa el marcador del fitxer al començament del
  // registre a modificar
  fseek(f, desp, SEEK_SET);
  // Llegeix el registre a modificar
  fread(&reg, bytesPerReg, 1, f);
  if (ferror(f))
  {
```

```
      printf("Error en llegir un registre del fitxer.\n");
    }
    else
    {
      printf("Nom:     %s\n", reg.nom);
      printf("Adreça:  %s\n", reg.adreca);
      printf("Telèfon: %ld\n\n", reg.telefon);
    }
}

void modificaReg(FILE *f, int nreg)
{
  struct registre reg;  // variable de tipus registre
  int bytesPerReg = sizeof(struct registre);

  printf("Nom:     "); gets(reg.nom);
  printf("Adreça:  "); gets(reg.adreca);
  printf("Telèfon: "); scanf("%ld", &reg.telefon);
  fflush(stdin);
  // Escriure un registre en el fitxer
  // fseek(f, -bytesPerReg, SEEK_CUR);
  fseek(f,bytesPerReg*(nreg-1),SEEK_SET);
  fwrite (&reg, bytesPerReg, 1, f);
  if (ferror(f))
    printf("Error en escriure un registre en el fitxer.\n");
}
```

Apèndix: Biblioteca de problemes

Exercicis bàsics

1. Construïu una aplicació que es pugui cridar des de la línia de comandaments de MS-DOS amb paràmetres i que mostri per pantalla la quantitat de paràmetres passats i quins són. Per exemple:

```
c:\>est1 Francesc Xavier Vallès Vegas

El nombre total d'arguments és 5
L'argument 0 és c:\>est1.exe
L'argument 1 és Francesc
L'argument 2 és Xavier
L'argument 3 és Vallès
L'argument 4 és Vegas
```

2. Construïu una aplicació que es pugui cridar des de la línia de comandaments de MS-DOS amb paràmetres numèrics i que mostri per pantalla la mitjana de tots els nombres. Per exemple:

```
c:\>est2 2 3.4 5 6.25

Mitjana de les 4 dades: 4.1625
```

3. Construïu una aplicació que es pugui cridar des de la línia de comandaments de MS-DOS amb dos paràmetres: un caràcter i un enter. L'aplicació ha d'escriure el caràcter indicat pel primer argument el nombre de vegades que indiqui el segon argument. Per exemple:

```
c:\>est3 x 10

xxxxxxxxxx
```

4. Utilitzant les funcions `fgetc()` i `fputc()` i ajudant-vos només d'una variable caràcter per a llegir i escriure, realitzeu els següents programes:
   a) Programa que llegeixi des del teclat una línia de text. Seguidament, l'ha d'escriure en un arxiu.
   b) Programa que llegeixi des de l'arxiu anterior una línia de text i que la mostri per pantalla.

5. Utilitzant les funcions `fgets()` i `fputs()` i ajudant-vos d'una variable cadena de 81 caràcters per a llegir i escriure, realitzeu els següents programes:
   a) Programa que llegeixi des del teclat un text de vàries línies. Seguidament, l'ha d'escriure en un arxiu.
   b) Programa que llegeixi des de l'arxiu anterior un text de vàries línies i el mostri per pantalla.

6. Escriviu un programa que emuli el comandament *copy* del MS-DOS. Utilitzeu les funcions `fgetc()` i `fputc()`. Ajudeu-vos només d'una variable caràcter per a llegir i escriure.

7. Modifiqueu el programa de l'agenda del tema anterior, de forma que permeti guardar les dades en disc i recuperar-les.

8. Afegiu al programa de l'agenda una funció que generi un fitxer de text en què apareguin en cada línia les dades d'un contacte. Optativament ha de permetre escriure aquestes línies directament en la impressora.

9. Escriviu un programa que rebi el nom d'un fitxer a través de la línia de comandament i trobi la mida d'aquest fitxer. Utilitzeu les funcions `fseek()` i `ftell()`.

10. Realitzeu un programa que divideixi un arxiu en dues parts iguals. Feu també una altra versió per a tornar a ajuntar-los. Els noms dels fitxers origen i destinació s'han de passar des de la línia de comandaments.

Exercicis avançats

11. Feu una aplicació que faci les parts d'un fitxer que siguin necessàries per tal que cadascuna de les parts càpiga en un disc de 3 ½ polzades. Per exemple, si l'arxiu original té una mida de 5.356.320 bytes, s'han de fer 4 parts, tres de 1.450.000 i una de 1.006.320 bytes. Feu que el nom de l'arxiu original s'introdueixi com un argument de la funció `main()`. El noms de les parts serà arxiu1.dat, arxiu2.dat,.... Implementeu també l'opció inversa. El programa es podrà executar amb aquestes dues opcions:

```
nom_programa –p fitxer_origen
nom_programa –u fitxer_unió fitxer_origen1 ... fitxer_origenN
```

12. Escriviu diversos programes (o només un, amb un menú) que treballin amb el nom, cognom, edat i nota de diversos alumnes, utilitzant la següent estructura:

```
struct fitxa {
   char  nom[20];
   char  cognom[20];
   int   edat;
   float nota;
}
```

    a) Programa que llegeixi des del teclat les dades de diversos alumnes i les escrigui en un arxiu de text. Utilitzeu la funció `fprintf()`.
    b) Programa que llegeixi des de l'arxiu anterior les dades de diversos alumnes i les mostri per pantalla. Utilitzeu la funció `fscanf()`.
    c) Programa que llegeixi des del teclat les dades de diversos alumnes i les escrigui en un arxiu binari. Utilitzeu la funció `fwrite()`.
    d) Programa que llegeixi des de l'arxiu anterior les dades de diversos alumnes i les mostri per pantalla. Utilitzeu la funció `fread()`.
    e) Programa que a partir de l'arxiu de l'apartat a) en creï un com el de l'apartat c).
    f) Programa que a partir de l'arxiu de l'apartat c) en creï un com el de l'apartat a).

13. Feu una aplicació on es declari un vector de tipus `double` de 100 elements. L'aplicació ens ha de permetre introduir aquests números per teclat fins a un màxim de 100. Amb la introducció d'un -1 es finalitzarà la introducció de dades. El número -1 no s'ha de guardar. Una vegada introduïts tots aquests números, s'han de guardar en disc en una sola operació amb la funció `fwrite()`. El nom de l'arxiu on es guardaran tots aquests números també s'introduirà per teclat abans d'introduir qualsevol dada. Feu un altre programa que permeti recuperar les dades del fitxer i imprimir-les a la pantalla.

14. Escriviu un programa que rebi el nom d'un fitxer de text a través de la línia de comandament i li canviï totes les minúscules per majúscules.

15. Escriviu un programa que treballi amb el nom, cognom, edat i nota de diversos alumnes, utilitzant la següent estructura (no heu d'utilitzar cap vector d'estructures) i que disposi de les següents funcions:

```
struct fitxa {
```

```
    int   codi;
    char  nom[20];
    char  cognom[20];
    float nota;
}
```

a) Obre base de dades: ha de crear un fitxer nou de dades o obrir un existent.
b) Insereix alumnes: ha d'inserir al final del fitxer noves dades d'alumnes. El programa s'ha d'assegurar que el codi de l'alumne nou no existeix.
c) Mostra alumnes: ha de mostrar per pantalla les dades de tots els alumnes.
d) Cerca nom: ha de cercar i mostrar les dades de l'alumne que tingui el nom introduït.
e) Modifica alumne: ha de preguntar per un codi d'alumne i deixar a l'usuari la possibilitat de modificar les seves dades.
f) Elimina alumne: a partir del codi d'un alumne ha d'esborrar aquest de la base de dades. Per a esborrar-lo es posarà el seu codi a 0.
g) Compacta base de dades: ha d'esborrar definitivament del fitxer les dades dels alumnes eliminats.
h) Tanca base de dades: ha de tancar el fitxer de dades.

16. Afegiu al programa anterior les següents funcions:
    a) Exportar a Excel: ha de permetre exportar les dades del fitxer a l'Excel per a poder tractar-les allí i fer un gràfic estadístic. L'Excel entendrà el format si és de tipus text, cada fila està separada per un canvi de línia i cada camp per algun caràcter especial (utilitzeu el tabulador).
    b) Importar d'Access: disposem d'una taula en Access on tenim totes les dades dels alumnes. L'hem exportat a format text, i les files han quedat delimitades per un canvi de línia, i les columnes per un punt i coma. Volem importar aquestes dades a la nostra base de dades. A més, tenim un problema previ: el nostre ordinador té configurat com a idioma el català, per tant l'Access exporta els nombres reals separant la part fraccionària per una coma. Primer haurem de convertir totes les comes a punts abans d'intentar la importació.

17. Es vol escriure una aplicació per a comprimir i descomprimir fitxers binaris amb informació gràfica (fitxers `bmp`).

    Cada fitxer gràfic es composa d'un conjunt de bytes. Cada byte és un valor de 0 a 255 que representa el nivell de gris d'un píxel del gràfic.

    L'algorisme de compressió és el següent: "cada seqüència d'un o més bytes *del mateix valor* que hi hagi al fitxer origen es va a representar amb dos bytes, de tal forma que el primer representa el nivell de gris llegit del fitxer origen i el segon el nombre total de bytes que hi ha a la seqüència". Si la seqüència és superior als 255 bytes, utilitzeu repetidament la representació de dos bytes que siguin necessàries.

    Per a implementar aquesta aplicació es demana:

    a) Escriviu una funció que generi un fitxer comprimit a partir d'un fitxer gràfic, en el que cada registre sigui de la forma:

```
struct registre
{
   unsigned char pixel;
   unsigned char total_bytes;
```

```
}
```

El tipus de dades `char`, tot i que se sol tractar com si el seu rang fos 0..255, i normalment no dóna cap problema, certament és -128..127. Per a assegurar-nos que el seu rang és 0..255 cal definir-lo com a `unsigned char`.

El prototip d'aquesta funció és:

```
void comprimir (FILE *forigen, FILE *fcomprimit);
```

on `forigen` és el flux que referencia al fitxer gràfic que desitgem comprimir i `fcomprimit` és el flux que referencia al fitxer on emmagatzemarem les dades comprimides.

b) Escriviu una funció amb el prototip següent:

```
void descomprimir (FILE *fcomprimir, FILE *fdestinacio);
```

Aquesta funció descomprimirà les dades que hi ha al fitxer referenciat per `fcomprimit` i les dipositarà en el fitxer referenciat per `fdestinacio`.

c) Escriviu un programa de nom `compdesc` que pugui ser utilitzat de la forma següent:

```
compdesc –c forigen fdestinacio        per comprimir
compdesc –d forigen fdestinacio        per descomprimir
```

18. Volem realitzar un programa per al tractament de les factures d'una determinada empresa. Les factures de l'empresa tenen el següent format:

```
Número Factura: xxxx
Data: xx/xx/xxxx
Número Client: xxxxx

Codi producte   Descripció    Quantitat    Preu      Total
xxxx            xxxxxx       xxxxx        xxxxx     xxxxx
...
xxxx            xxxxxx       xxxxx        xxxxx     xxxxx

                                          Total a pagar          xxxxx
```

L'empresa disposa d'una taula de productes (sabem que mai hi hauran més de 100), amb la seva descripció i preu, que s'emmagatzema en un fitxer segons la següent estructura:

```
struct prod
{
   int id_producte;
   char descripció[40];
   float preu;
};
```

Les factures s'emmagatzemen en un fitxer, on es desen dos tipus d'estructures:

```
struct cap
{
   int numfact;
   struct data datafact;
   int idclient;
   int num_linies;
```

```
};
struct linia
{
   int id_producte;
   int quantitat;
};
```

Cada factura del fitxer consta d'una estructura `cap` seguida de tantes estructures `linia` com línies tingui la factura. Per exemple:

1	12/01/2003	12	3	1	5	4	7	2	2	2	12/01/2003	7	2	3	2	4	3		

Implementeu un programa que realitzi el tractament de les factures de l'empresa. Com a mínim ha de permetre entrar productes, i entrar i visualitzar factures amb el format adequat. Podeu ampliar-lo amb més opcions: visualitzar i esborrar productes, esborrar factures, realitzar recerques, tractar també dades dels clients, etc.