Ficheros Binarios

Eva Lucrecia Gibaja Galindo
Dpto. Informática y Análisis Numérico

Introducción

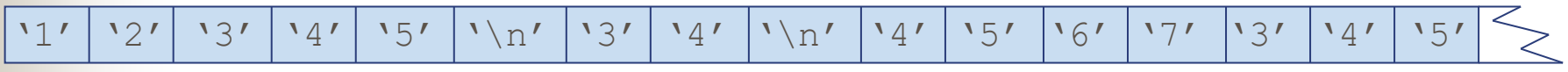
- **Ejemplo**. Supongamos una aplicación que debe guardar en un fichero una serie de *n* números enteros positivos (representados como `unsigned int`). Supongamos que *n=5* y se genera la siguiente secuencia:

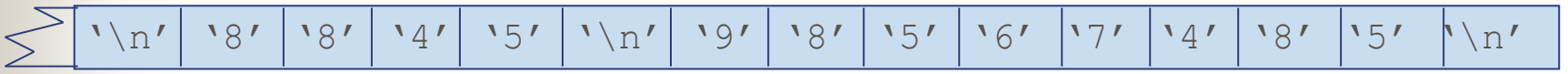
```
12345, 34, 4567345, 8845, 98367485
```

- Esta serie puede almacenarse en formato texto o binario

Introducción

1. ***Texto***. Almacenar cada entero en una línea del fichero o separados por un espacio o tabulador (indiferente)
   a) Si `sizeof(unsigned int)` es 4, el máximo entero sin signo que puede utilizarse es 4294967295→10 cifras (`limits.h`)
   b) Si los datos se generan de forma uniforme podemos tener valores desde 1 a 10 cifras con la misma probabilidad a priori
   c) Cada número ocupa tanto espacio como cifras tiene
   d) El tamaño del fichero es la suma del total de cifras de los números y separadores separadores: ***31 bytes: 26 cifras y 5 separadores***





Introducción

2. ***Binario***. Cada valor se almacena según su *representación interna* (copia exacta de como está en memoria)
   a) Cada valor se almacenaría exactamente con 4 *bytes*
   b) No se necesitan separadores para delimitar cada valor
   c) El tamaño del fichero será de ***20 bytes: 4 bytes/dato 5 datos***

Salvo en circunstancias muy extrañas, la adopción del formato binario para estos ficheros de datos resulta más ventajosa

Introducción

- **Fichero binario**. Sucesión de *bytes*. Más general que los ficheros de texto
- **Registro activo**. Aquel que va a ser procesado en la siguiente operación con el fichero
- **Cursor**. Marca interna que siempre apunta la registro activo en cada momento. El cursor se incrementa automáticamente cada vez que se procesa un registro (se lee o se escribe)
- **Clave**. Valor (o un conjunto de valores) de un registro que lo identifica unívocamente. Es diferente para todos los registros

Creación y apertura de ficheros binarios

```
FILE* fopen (const char* nombre, const char*modo);
```

- Abre el fichero físico cuyo nombre está referenciado por `nombre` y devuelve un puntero a `FILE` ó `NULL` si hay algún problema durante su apertura. La cadena `modo` puede ser:

Modo	Acciones	Cursor	Si existe	Si no existe
rb	Lectura	Inicio	Abre	Código error
wb	Escritura	Inicio	Borra contenido	Crea
ab	Adición	Final	Abre. Agrega al final	Crea
r+b	L/E	Inicio	Abre. Agrega al inicio, sobreescribiendo	Código error
w+b	L/E	Inicio	Borra contenido	Crea
a+b	L/Adición	Final	Abre. Agrega al final	Crea

E/S de ficheros binarios

```
size_t fread(void* ptr, size_t tam, size_t num, FILE* f);
```

- Lee un máximo de `num` objetos de tamaño `tam` del fichero referenciado por `f` y los deja en el *buffer* referenciado por `ptr`
- Devuelve el número de objetos leídos, que puede ser menor que `num` si hay algún error o se alcanza el fin del fichero
- El cursor del fichero avanza los *bytes* leídos

```
size_t fwrite(const void* ptr,size_t tam,size_t num, FILE* f);
```

- Escribe en el fichero referenciado por `f` un máximo de `num` objetos de tamaño `tam` tomados del *buffer* referenciado por `ptr`
- Devuelve el número de objetos escritos, que puede ser menor que `num` si hay algún error
- El cursor del fichero avanza los *bytes* escritos

copia.c

E/S de ficheros binarios. Ejemplo

```
#include <stdio.h>
#define TAM_BUF 1024 //Tamanio del buffer en bytes
typedef char byte;
int main()
{ FILE *fOrigen, *fDestino;
  char nomOrigen[]="origen.doc", nomDestino[]="destino.doc";
  long leidos;
  byte buffer[TAM_BUF];
  if((fOrigen=fopen(nomOrigen, "rb"))==NULL)
  { printf("\nError: no pudedo abrir %s", nomOrigen);exit(-1);}
  if((fDestino=fopen(nomDestino, "wb"))==NULL)
  { printf("\nError: no pudedo crear %s", nomDestino); exit(-1);}
```

copia.c

E/S de ficheros binarios. Ejemplo

```
//En cada iteracion intentamos leer TAM_BUF objetos
//de 1 byte. Escribimos lo que hemos leido
while((leidos=fread(buffer,sizeof(byte),TAM_BUF, fOrigen))>0)
{
   fwrite(buffer, 1, leidos, fDestino);
}
fclose(fOrigen);
fclose(fDestino);
return(0);
}
```

nº de elementos

tamaño

sizeof() nos dice el tamaño de un tipo de dato

Como no conocemos la estructura del fichero, leeremos en bloques de TAM_BUF bytes

TAM_BUF	TAM_BUF	TAM_BUF	leidos

Funciones de posicionamiento

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

- Devuelve:
  - El número de *bytes* desde el principio hasta la posición actual del cursor
  - `-1L` si hay error

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

- Devuelve: *cierto* (≠ 0) si se produce algún error
- Establece la posición actual del cursor
  - Ficheros binarios: la nueva posición se establece como un desplazamiento de `desp` *bytes* a partir de `origen`, que puede ser:
    - `SEEK_SET` / 0 : inicio del fichero
    - `SEEK_CUR` / 1: posición actual del cursor
    - `SEEK_END` / 2 : fin del fichero
  - Ficheros de texto: `desp` debe ser:
    - 0L
    - Un valor devuelto por `ftell()`. En este caso, `origen` será `SEEK_SET`

```
fseek(f, 0L, SEEK_SET);
fseek(f, 0L, SEEK_END);
fseek(f, n*sizeof(struct cliente), SEEK_SET);
fseek(f, 2*sizeof(struct cliente), SEEK_CUR);
fseek(f, -1*sizeof(struct cliente), SEEK_CUR);
fseek(f, -1*sizeof(struct cliente), SEEK_END)
```

clientes.c

Numero de registros de un fichero binario

Para ficheros de cualquier tipo!!

```
long tamanio(char* nombreFichero)
{
  FILE* f;
  long tam;
  if((f=fopen(nombreFichero, "rb"))==NULL)
  {
    fprintf(stderr, "\nError: no puedo abrir el fichero <%s>",nombreFichero);
    exit(-1);
  }
  if(fseek(f, 0L, SEEK_END))
  {
    fprintf(stderr, "\nError: no puedo usar el fichero <%s>", nombreFichero);
    exit(-1);
  }
  tam = ftell(f);
  fclose(f);
  return tam;
}
```

$$nº\ registros = \frac{bytes\ en\ fichero}{bytes\ /\ registro}$$

$$1Kbyte = 2^{10} bytes = 1024 bytes$$

ftell() devuelve el numero de bytes desde el principio del fichero hasta la posición actual del cursor

```
int main(int argc, char* argv)
{
    char nombreFichero[MAX_LINE];
    printf("\nIndique el nombre del fichero:");
    scanf("%s", nombreFichero);
    tam = tamanio(nombreFichero);
    printf("\nFichero:<%s>", nombreFichero);
    printf("\n%ld bytes", tam);
    printf("\n%5.2f Kbytes", tam/1024.0);
    printf("\n\tRegistros: %d", tam/sizeof(struct cliente));
    return 0;
}
```

Otras funciones para ficheros de texto y binarios

```
int remove (const char* nombre);
```

- Borra el fichero llamado nombre
  - Devuelve un valor distinto de cero en caso de error

```
int rename (const char* viejo, const char* nuevo);
```

- Cambia el nombre del fichero llamado `viejo` por el de `nuevo`
  - Devuelve un valor distinto de cero en caso de error

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

- Fuerza a que el fichero `f` se libere: se vacía el *buffer* asociado al fichero de salida. El efecto está indefinido para ficheros de entrada. Devuelve `EOF` si hay algún error de escritura y cero en otro caso
  - `fflush(NULL)` libera todos los ficheros de salida
  - `fflush(stdout)` libera el *buffer* del dispositivo de salida
  - **`fflush(stdin)` su efecto está indefinido → NO UTILIZARLO**

Procesamiento de un fichero

- Lectura de un fichero
  - Lectura elemento a elemento
    - Listar registros de un fichero
    - Búsqueda en un fichero
  - Lectura en bloque
    - Fichero a vector
- Escritura en un fichero
  - Crear fichero vacío
  - Añadir datos al final de un fichero
- Posicionamiento del cursor
  - Número de registros / *bytes*
  - Ver registro i-ésimo
- Modificación de los datos de un registro
  - Actualización
    - Borrado lógico → actualización
  - Borrado físico

datosPersonalesB1.c

Lectura elemento a elemento

Listar fichero

```
struct DatosPersonales
{
  //clave unica del registro
  long dni;
  char nombre[MAX_LINEA];
  char apellido[MAX_LINEA];
  float salario;
};
```

```
void verFichero(char* nombreFichero)
{
 FILE *pFichero;
 struct DatosPersonales persona;

 pFichero = fopen(nombreFichero, "rb");

 /* lee datos del fichero de uno en uno hasta que llega al final */
while(fread(&persona,sizeof(struct DatosPersonales),1,pFichero)==1)
 {
    /* Muestra el registro por pantalla */
    escribirDatosPersonales(persona);
 }
 fclose(pFichero);
}
```

Lectura en bloque

Fichero a vector

```
struct DatosPersonales* ficheroAVector(char* nombreFichero, long* nEle)
{
   struct DatosPersonales* V;
   FILE* pFichero;

   *nEle = contarRegistros(nombreFichero);
   V = reservarVector(*nEle);
   pFichero = fopen(nombreFichero, "rb");

   /*Leemos todo el fichero (mas rapido que elemento a elemento)*/
   fread(V, sizeof(struct DatosPersonales),*nEle, pFichero);
   fclose(pFichero);
   return(V);
}
```

Vector a fichero

Casos:

1. El fichero está recién creado ⇒ No tenemos que preocuparnos por el cursor, situado en la cabecera del fichero
2. El fichero está creado y contiene datos ⇒ Para conservar la información existente hay que añadir los nuevos datos al final

clientes.c

Vector a fichero. Sobreescribir

```
void clientesAFichero(char* nombreFichero, struct cliente
    Clientes[], int tope)
{
  FILE* f;
  if((f=fopen(nombreFichero, "wb"))==NULL)
  {
    fprintf(stderr, "\nError: no se puede abrir <%s>",
    nombreFichero);
  }
  else
  {
    //Escribe todos los clientes
    fwrite(Clientes, sizeof(struct cliente), tope, f);
    printf("\n\tFichero <%s> salvado a disco", nombreFichero);
    fclose(f);
  }
}
```

```
struct cliente
{ char DNI[MAX_DNI];
  int cuenta;
  char nombre[MAX_NOMBRE];
  float saldo;
};
```

Escribimos todo el vector con un solo fwrite

clientes.c

Esta función es igual que clientesAFichero(), sólo cambia el modo de apertura

Vector a fichero. Añadir

```
void addClientes(char* nombreFichero, struct cliente Clientes[],
    int tope)
{
  FILE* f;
  if((f=fopen(nombreFichero, "ab"))==NULL)
  {
    fprintf(stderr, "\nNo se puede abrir <%s>", nombreFichero);
  }
  else
  {
    //Escribe todos los clientes válidos del vector
    fwrite(Clientes, sizeof(struct cliente), tope, f);
    fclose(f);
  }
}
```

Escribimos todo el vector con un solo fwrite

Búsqueda en un fichero

```
int mostrarporNombre(char* nombreFichero, char *auxNombre)
{
 FILE *pFichero;
 struct DatosPersonales auxiliar;
 int encontrado = 0;
  /* abre fichero para lectura */
 pFichero = fopen(nombreFichero, "rb");
 while(fread(&auxiliar,sizeof(struct DatosPersonales),1,pFichero) == 1)
  {
   if (strcmp(auxiliar.nombre, auxNombre)==0)
   /* se ha encontrado un registro con ese nombre */
    {
     escribirDatosPersonales(auxiliar); /* se escriben sus datos */
     encontrado = 1;
    }
  }
 fclose(pFichero); /* se cierra el fichero */
 return encontrado;
}
```

Ver registro i-ésimo

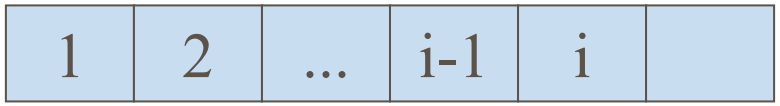
```
struct DatosPersonales registro_i(char* nombreFichero, long i)
{
 FILE *pFichero;
 struct DatosPersonales aux;

 /* Se abre para lectura el fichero */
 pFichero = fopen(nombreFichero, "rb");

 /* Nos desplazamos al final del registro i-1, desde el principio
 del fichero con la funcion fseek*/
 fseek(pFichero, (i-1)*sizeof(struct DatosPersonales), SEEK_SET);

 /* Se lee el siguiente registro */
 fread(&aux,sizeof(struct DatosPersonales),1,pFichero);

 fclose(pFichero);
 /* devuelve el registro leido */
 return aux;
}
```



Reg_1 Reg_2 ... Reg_{i-1} ...

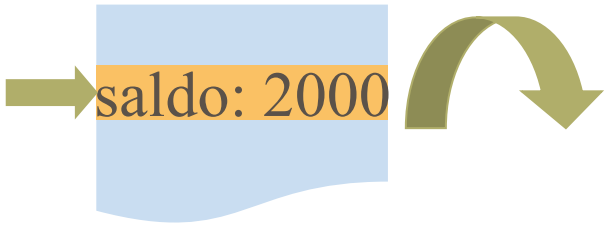
i estará en [1..num_registros] y no supera el número de registros del fichero

Modificar un registro

1. Localizar en el fichero el registro a modificar
   1. Situar el cursor al principio del fichero
      - Abriendo el fichero
      - Con una función específica
   2. Leer registro a registro el fichero y almacenándolos en una variable auxiliar
   3. Comparar con el contenido de algún campo (generalmente el campo clave)
2. Una vez encontrado el registro, realizar las modificaciones sobre la variable auxiliar
3. Escribir la variable auxiliar en el fichero en la misma posición en que estaba
   - Para ello hay que retroceder el cursor, pues la lectura ha avanzado el cursor

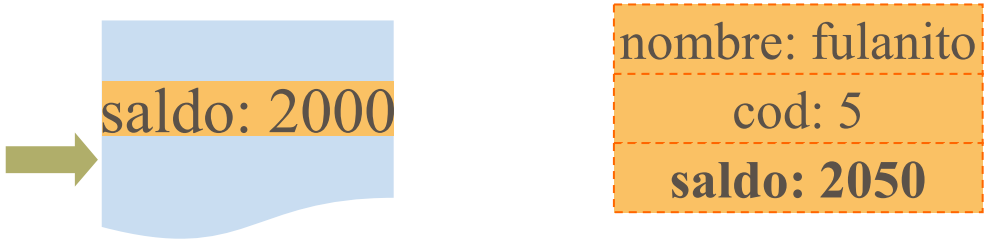
Modificar un registro

1. Localizar el registro



copiar en registro auxiliar (avanza el cursor!!)

2. Modificar registro auxiliar



3. Retoceder el cursor

4. Escribir registro

Modificar un registro

```
void modificaSaldo(char* nombreFichero, char* dniCliente, float nuevoSaldo)
{ FILE* f;
  struct cliente cliAux;
  int encontrado=0;

  if((f=fopen(nombreFichero, "r+b"))==NULL)
  {fprintf(stderr, "Error: no se puede abrir <%s>", nombreFichero);}
  else{
    while((fread(&cliAux, sizeof(struct cliente), 1, f)==1)&&(!encontrado)){
      if(strcmp(cliAux.DNI, dniCliente)==0){
        cliAux.saldo = nuevoSaldo; //Actualizamos registro local
        fseek(f, -(int)sizeof(struct cliente), SEEK_CUR); //Retrocedemos posicion
        fwrite(&cliAux, sizeof(struct cliente), 1, f); //Escribimos registro actualizado
        fflush(f); //el estandar requiere fflush para hacer fread despues de write
        encontrado = 1;
      }}
  fclose(f);
  if(!encontrado)  {printf("\nNo existe el cliente: <%s>", dniCliente);}
}
```

Borrado físico *vs*. borrado lógico

- **Borrado lógico**. No es un borrado real del registro. Es una **actualización** que marca, de alguna forma los registros que ya no son válidos y utilizar esos huecos para futuras inserciones
- Formas de marcar:
  - Si existe un campo clave dándole un valor nulo: NULL, cadena vacía, etc. Problema si no es posible definir un valor nulo
  - Añadir a cada registro un campo adicional de tipo lógico borrado tal que si su valor es cierto, entendemos que el registro está borrado. Desventaja, cada registro tiene un campo adicional que aumenta el tamaño del fichero

Borrado físico *vs*. borrado lógico

- **Borrado físico**. Eliminar un registro de un fichero para que el espacio que éste ocupa en el disco quede libre. Dos formas:
  1. Para borrar el registro de la posición i de un fichero de n registros, ir desplazando los registros de las posiciones i+1, i+2, ... n a una posición menos, de manera que el de la posición i+1 quede grabado sobre el de la posición i, etc. Muy costoso en tiempo. Si queremos borrar varios registros hay que repetir el proceso para todos
  2. Copiar en un nuevo fichero sólo los registros que se desee conservar. Inconveniente, necesidad de espacio en disco, pero es más eficiente
- Se realiza cuando los sistemas de la empresa no están siendo utilizados (durante la noche)
- Se borran a la vez todos los registros marcados para borrar

Borrado lógico

```
void borradoLogico(char* nombreFichero, char* dniCliente)
{ FILE* f;
  struct cliente cliAux;
  int encontrado=0;
  f=fopen(nombreFichero, "r+b");
  while((fread(&cliAux, sizeof(struct cliente), 1, f)==1)&&(!encontrado))
  {   if(strcmp(cliAux.DNI, dniCliente)==0)
      {
        strcpy(cliAux.DNI,""); //cadena vacia
        printf("\n\tdniNuevo: %s ", cliAux.DNI);
        fseek(f, -(int)sizeof(struct cliente), SEEK_CUR);
        fwrite(&cliAux, sizeof(struct cliente), 1, f);
        fflush(f);
        encontrado = 1;
      }
   }
   fclose(f);
}
```

retroceso del cursor

clientes.c

Borrado físico

Si el campo tiene la marca de borrado (dni es “”), se borrará

El borrado físico requiere dos ficheros

```
void borradoFisico(char* nombreFichero)
{ FILE* f, *faux;
  struct cliente cliAux;
  int encontrado=0;
  f=(FILE*)fopen(nombreFichero, "rb");
  faux=(FILE*)fopen("tmp"     , "wb");
  while(fread(&cliAux, sizeof(struct cliente), 1, f)==1)
  { if(strcmp(cliAux.DNI, "")!=0)
      {fwrite(&cliAux, sizeof(struct cliente), 1, faux);}
      else
      {encontrado=1;}
  }//while
  fclose(f);
  fclose(faux);
  if(!encontrado)
  { remove("tmp");} //No hay registros para borrar
  else
  { remove(nombreFichero);
      rename("tmp", nombreFichero);
  }
}
```

Resumen

Ficheros de texto y binarios	Apertura	fopen
	Cierre	fclose
	Otras	remove, rename, ~~fflush~~, ~~feof~~
Ficheros de texto	L/E de caracteres	fgetc, ~~getc~~, fputc, ~~putc~~
	L/E Líneas	fgets, fputs
	L/E Formato	fscanf, fprintf
Ficheros binarios	L/E	fread, fwrite
	Posicionamiento	fseek-ftell

Resumen

- Lectura de un fichero binario
  - Leer todo el fichero de una vez

```
*tope = fread(Clientes, sizeof(struct cliente), MAX_CLIENTES, f);
```

  - Leer elemento a elemento

```
while(fread(&clienteAux, sizeof(struct cliente), 1, f)==1){...}
```

  - Leer bloques de elementos

```
while((leidos=fread(buffer,sizeof(byte),TAM_BUF, fOrigen))>0){...}
```

Material adicional

Otras funciones de posicionamiento

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

- La traducción de `rewind` es *rebobinar*, esto es, *colocarse al principio del fichero*
  - Equivalente a `fseek(f, 0L, SEEK_SET);`

```
int fgetpos (FILE* f, fpos_t* ptr);
```

- Asigna la posición actual del fichero referenciado por `f` a `*ptr`. El tipo `fpos_t` es adecuado para registrar tales valores. Devuelve un valor distinto de cero si hay error

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

- Establece la posición actual del fichero referenciado por `f` con el valor dado por `*ptr`. Devuelve un valor distinto de cero si hay error
  - Las funciones *fgetpos* y *fsetpos* son interfaces alternativas equivalentes a *ftell* y *fseek* (con el origen puesto a SEEK_SET). Utilizar mejor *ftell* y *fseek*

Otras Funciones

```
FILE* freopen(const char* nbre,const char* modo,FILE* f);
```

- Cierra el fichero asociado con *f* y abre el fichero llamado `nbre` en el modo especificado por `modo` y lo asocia con el fichero `f`. Devuelve `FILE*` ó `NULL`

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

- Devuelve un valor distinto de 0 si se ha producido un error durante la última operación sobre el archivo

```
FILE* tmpfile();
```

- Crea un fichero temporal (en modo "wb+") que se borra al ser cerrado o cuando termina el programa normalmente. Devuelve FILE * si todo va bien ó NULL en caso de error

```
char* tmpnam(char s[L_tmpnam]);
```

- Genera una cadena de caracteres que es un nombre válido para ficheros y que no es igual al nombre de un fichero existente. La función *tmpnam* genera una cadena diferente cada vez que es llamada

La Función `feof()`

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

- Devuelve:
  - *falso* (0) si el cursor no ha sobrepasado el final de fichero
  - *cierto* (≠ 0) si el cursor ha sobrepasado el fin del fichero
- Requiere realizar **lectura anticipada** lo veremos en los ejemplos
- Una forma más general de comprobar si se ha alcanzado el fin del fichero

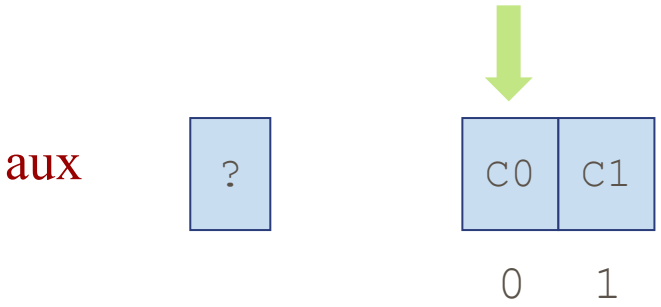
Fichero a vector (3). Con feof()

```
void clientesAVector3(char* nombreFichero,struct cliente Clientes[], int* tope)
{ FILE* f;
  int cuenta =0;
  struct cliente clienteAux;
  if((f=fopen(nombreFichero, "rb"))==NULL)
  {fprintf(stderr, "\nNo se puede abrir <%s>", nombreFichero);}
  else
  { *tope=0;
    cuenta = fread(&clienteAux, sizeof(struct cliente), 1, f);
    while((!feof(f))&&(*tope<MAX_CLIENTES))
    { //El cliente se pasa al vector si no tiene marca de borrado
      if(strcmp(clienteAux.DNI, "")!=0)
      { Clientes[*tope]=clienteAux;
        *tope = *tope+cuenta;}
      //Leemos el siguiente cliente
      cuenta = fread(&clienteAux, sizeof(struct cliente), 1, f);
    }
    fclose(f);
  }
}
```

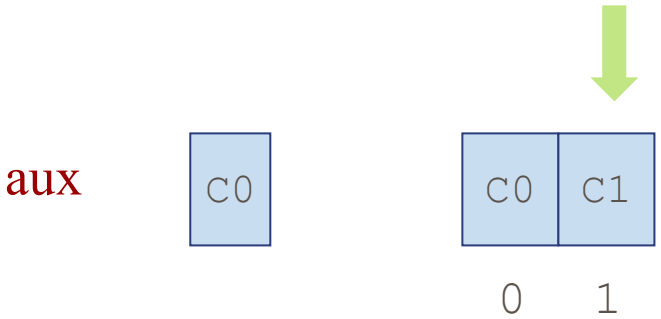
lectura anticipada

Fichero a vector (3). Con feof()

1. Apertura del fichero en modo lectura

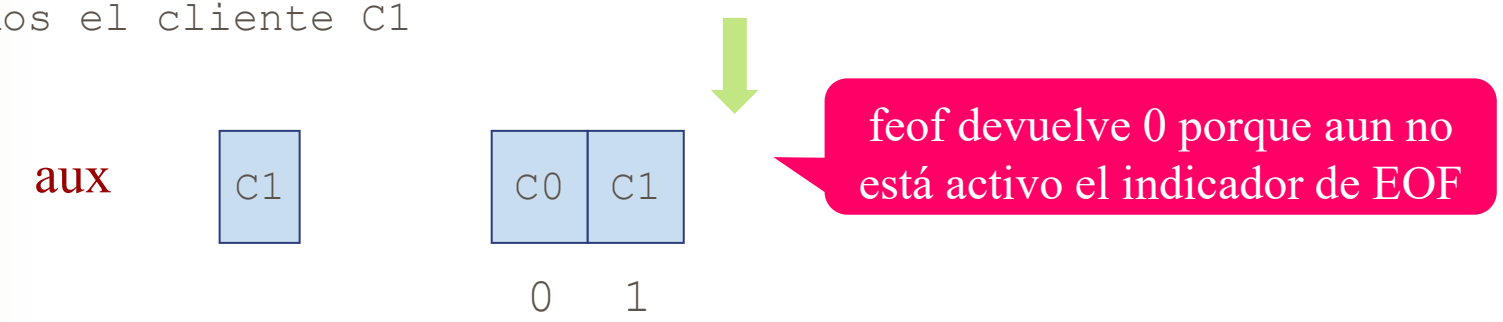


2.Lectura anticipada, leemos el cliente C0

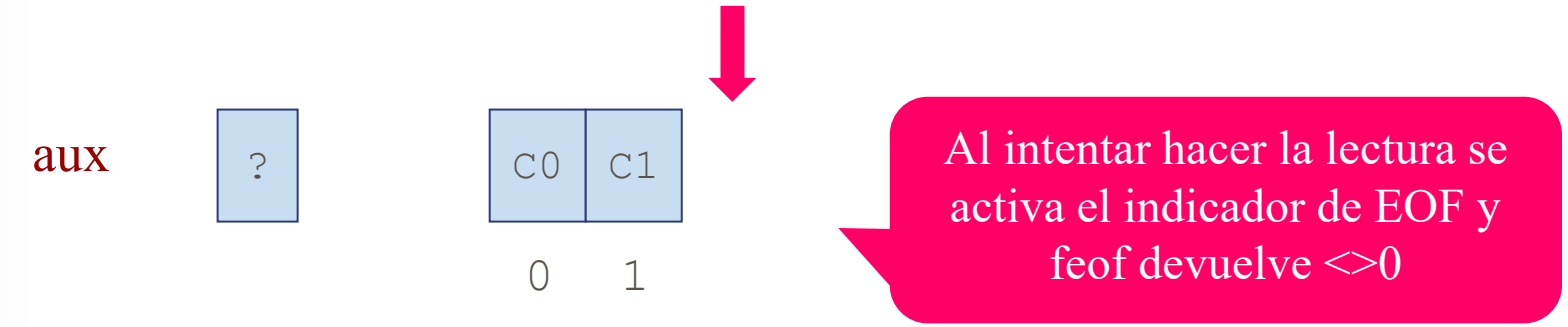


Fichero a vector (3). Con feof()

3. Leemos el cliente C1



4. Intento de lectura, se activa el indicador de fin de fichero



clientes.c

Búsqueda en un fichero. Con feof()

```
void listaSaldoSuperior2(char* nombreFichero, float saldoTope)
{ FILE* f;
  struct cliente cliAux;
  if((f=fopen(nombreFichero, "rb"))==NULL)
  {fprintf(stderr, "Error: no se puede abrir <%s>", nombreFichero);}
  else
  {
    printf("\n\nMODULO DE LISTADO DE CLIENTES CON SALDO >%d", saldoTope);
    fread(&cliAux, sizeof(struct cliente), 1, f);
    while(!feof(f))
    {
      if((cliAux.saldo>=saldoTope)&&(strcmp(cliAux.DNI, "")!=0))
      {escribeCliente(cliAux); }
      //Leemos el siguiente cliente
      fread(&cliAux, sizeof(struct cliente), 1, f);
    }
    fclose(f);
  }
}
```

lectura anticipada

clientes.c

Búsqueda en un fichero. Con feof()

```
void consultaSaldo2(char* nombreFichero, char* dniCliente)
{ FILE* f;
  struct cliente cliAux;
  int encontrado=0;
  if((f=fopen(nombreFichero, "rb"))==NULL)
  {fprintf(stderr, "Error: no se puede abrir <%s>", nombreFichero);}
  else
  { printf("\n\nMODULO DE CONSULTA DE SALDO:");
    fread(&cliAux, sizeof(struct cliente), 1, f);
    while(!feof(f))
    { if(strcmp(cliAux.DNI, dniCliente)==0)
      { escribeCliente(cliAux);
        encontrado = 1;
      }
      fread(&cliAux, sizeof(struct cliente), 1, f);
    }
    fclose(f);
    if(!encontrado)
    {printf("\nNo existe el cliente: <%s>", dniCliente);}
  }
}
```

lectura anticipada

clientes.c

Modificar un registro. Con feof()

```
void modificaSaldo2(char* nombreFichero, char* dniCliente, float nuevoSaldo)
{ FILE* f;
  struct cliente cliAux;
  int encontrado=0;
  if((f=fopen(nombreFichero, "r+b"))==NULL)
  {fprintf(stderr, "Error: no se puede abrir <%s>", nombreFichero);}
  else{
    fread(&cliAux, sizeof(struct cliente), 1, f);
    while(!feof(f)&&(!encontrado)){
      if(strcmp(cliAux.DNI, dniCliente)==0){
        cliAux.saldo = nuevoSaldo;
        fseek(f, -(int)sizeof(struct cliente), SEEK_CUR);
        fwrite(&cliAux, sizeof(struct cliente), 1, f);
        fflush(f);encontrado = 1;}
      fread(&cliAux, sizeof(struct cliente), 1, f);}
    fclose(f);
    if(!encontrado)
    {printf("\nNo existe el cliente: <%s>", dniCliente);}}}
```

lectura anticipada

Borrado lógico. Con feof()

```
void borradoLogico2(char* nombreFichero, char* dniCliente)
{
  FILE* f;
  struct cliente cliAux;
  int encontrado=0;

  if((f=fopen(nombreFichero, "r+b"))==NULL)
  {
    fprintf(stderr, "\nNo se puede abrir <%s>", nombreFichero);

  }
  else
  {
    printf("\n\nMODULO DE BORRADO LOGICO POR DNI:");
    printf("\n-----------------------------------");
```

Borrado lógico. Con feof()

```
   fread(&cliAux, sizeof(struct cliente), 1, f);
   while((!feof(f))&&(!encontrado)){
      if(strcmp(cliAux.DNI, dniCliente)==0)
      { printf("\nDNI: %s", cliAux.DNI);
        printf("\n\tnombre: %s", cliAux.nombre);
        printf("\n\tcuenta: %d", cliAux.cuenta);
        printf("\n\tsaldo: %f ", cliAux.saldo);
        strcpy(cliAux.DNI,""); //cadena vacia
        printf("\n\tdniNuevo: %s ", cliAux.DNI);
        fseek(f, -(int)sizeof(struct cliente), SEEK_CUR);
        fwrite(&cliAux, sizeof(struct cliente), 1, f);
        fflush(f);
        encontrado = 1;
      }
      fread(&cliAux, sizeof(struct cliente), 1, f);
    }
    fclose(f);
    if(!encontrado){ printf("\nNo existe el cliente: <%s>", dniCliente);}
  }
}
```

lectura anticipada

retroceso del cursor

fgetsfeof.c
personas.txt

Ejemplo. Ficheros de texto y feof()

```
void leeVersion2(char* nombreFichero)
{
   FILE* fich;
   char cadena[30];

   if((fich=fopen(nombreFichero, "r"))==NULL)
     printf("\nNo se ha podido abrir el fichero <%s>", nombreFichero);
   else
   {
      fgets(cadena, 30, fich);
      while(!feof(fich))
      {
        if(cadena[strlen(cadena)-1]=='\n')
           cadena[strlen(cadena)-1]='\0';
        printf("<%s>\n", cadena);
        fgets(cadena, 30, fich);
      }
      if(cadena[0]!='\n')
         printf("<%s>\n", cadena);
      fclose(fich);
    }

}
```

Lectura anticipada

Si el fichero no termina en '\n' la ultima lectura contiene la última línea del fichero

Muestra2_1.c

Ejemplo.

```
void fileCopy(FILE* destino, FILE* fuente)
{ int c;
  while((c=getc(fuente))!=EOF)
  {putc(c, destino);}
}
```

Esta función sólo es válida para ficheros de texto.

```
void fileCopy(FILE* destino, FILE* fuente)
{ int c;
  c=getc(fuente); Lectura anticipada
  while(!feof(fuente))
  { putc(c, destino); //Escritura en destino
    c=getc(fuente);   //Nueva lectura
  }
}
```

Esta función es válida para ficheros de cualquier tipo, ya que la condición de terminación se evalúa utilizando feof().