CAPÍTULO 9
Archivos (ficheros)

9.1. Archivos y flujos (stream): La jerarquía de datos
9.2. Conceptos y definiciones = terminología
9.3. Soportes secuenciales y direccionables
9.4. Organización de archivos
9.5. Operaciones sobre archivos
9.6. Gestión de archivos
9.7. Flujos
9.8. Mantenimiento de archivos
9.9. Procesamiento de archivos secuenciales (algoritmos)
9.10. Procesamiento de archivos directos (algoritmos)
9.11. Procesamiento de archivos secuenciales indexados
9.12. Tipos de archivos: consideraciones prácticas en C/C++ y Java
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS
CONCEPTOS CLAVE
RESUMEN
EJERCICIOS

INTRODUCCIÓN

Los datos que se han tratado hasta este capítulo y procesados por un programa pueden residir simultáneamente en la memoria principal de la computadora. Sin embargo, grandes cantidades de datos se almacenan normalmente en dispositivos de memoria auxiliar. Las diferentes técnicas que han sido diseñadas para la estructuración de estas colecciones de datos complejas se alojaban en arrays; en este capítulo se realiza una introducción a la organización y gestión de datos estructurados sobre dispositivos de almacenamiento secundario, tales como cintas y discos magnéticos. Estas colecciones de datos se conocen como **archivos (ficheros)**. Las técnicas requeridas para gestionar archivos son diferentes de las técnicas de organización de datos que son efectivas en memoria principal, aunque se construyen sobre la base de esas técnicas. Este capítulo introductorio está concebido para la iniciación a los archivos, lo que son y sus misiones en los sistemas de información y de los problemas básicos en su organización y gestión.

9.1. ARCHIVOS Y FLUJOS (STREAM): LA JERARQUÍA DE DATOS

El almacenamiento de datos en variables y ***arrays*** **(arreglos)** es temporal; los datos se pierden cuando una variable sale de su ámbito o alcance de influencia, o bien cuando se termina el programa. La mayoría de las aplicaciones requieren que la información se almacene de forma persistente, es decir que no se borre o elimine cuando se termina la ejecución del programa. Por otra parte, en numerosas aplicaciones se requiere utilizar grandes cantidades de información que, normalmente, no caben en la memoria principal. Debido a estas causas se requiere utilizar **archivos (ficheros)** para almacenar de modo permanente grandes cantidades de datos, incluso después que los programas que crean los datos se terminan. Estos datos almacenados en archivos se conocen como **datos persistentes** y permanecen después de la duración de la ejecución del programa.

Las computadoras almacenan los archivos en dispositivos de almacenamiento secundarios, tales como discos CD, DVD, memorias *flash* USB, memorias de cámaras digitales, etc. En este capítulo se explicará cómo los programas escritos en un lenguaje de programación crean, actualizan o procesan archivos de datos.

El procesamiento de archivos es una de las características más importantes que un lenguaje de programación debe tener para soportar aplicaciones comerciales que procesan, normalmente, cantidades masivas de datos persistentes. La entrada de datos normalmente se realiza a través del teclado y la salida o resultados van a la pantalla. Estas operaciones, conocidas como Entrada/Salida (E/S), se realizan también hacia y desde los archivos.

Los programas que se crean con C/C++, Java u otros lenguajes necesitan interactuar con diferentes fuentes de datos. Los lenguajes antiguos como FORTRAN, Pascal o COBOL tenían integradas en el propio lenguaje las entradas y salidas; palabras reservadas como `PRINT`, `READ`, `write`, `writeln`, etc, son parte del vocabulario del lenguaje. Sin embargo, los lenguajes de programación modernos como C/C++ o Java/C# tienen entradas y salidas en el lenguaje y para acceder o almacenar información en una unidad de disco duro o en un CD o en un DVD, en páginas de un *sitio web* e incluso guardar bytes en la memoria de la computadora, se necesitan técnicas que pueden ser diferentes para diferente dispositivo de almacenamiento. Afortunadamente, los lenguajes citados anteriormente pueden almacenar y recuperar información, utilizando sistemas de comunicaciones denominados **flujos** que se implementan en bibliotecas estándar de funciones de E/S (en archivos de cabecera `stdio.h` y `cstdio.h`) en C, en una biblioteca estándar de clases (en archivos de cabecera `iostream` y `fstream`) en C++, o en el paquete Java.io en el lenguaje Java.

Las estructuras de datos enunciadas en los capítulos anteriores se encuentran almacenadas en la memoria central o principal. Este tipo de almacenamiento, conocido por *almacenamiento principal o primario*, tiene la ventaja de su pequeño tiempo de acceso y, además, que este tiempo necesario para acceder a los datos almacenados en una posición es el mismo que el tiempo necesario para acceder a los datos almacenados en otra posición del dispositivo —memoria principal—. Sin embargo, no siempre es posible almacenar los datos en la memoria central o principal de la computadora, debido a las limitaciones que su uso plantea:

- La cantidad de datos que puede manipular un programa no puede ser muy grande debido a la limitación de la memoria central de la computadora[1].
- La existencia de los datos en la memoria principal está supeditada al tiempo que la computadora está encendida y el programa ejecutándose (tiempo de vida efímero). Esto supone que los datos desaparecen de la memoria principal cuando la computadora se apaga o se deja de ejecutar el programa.

Estas limitaciones dificultan:

- La manipulación de gran número de datos, ya que —en ocasiones— pueden no caber en la memoria principal (aunque hoy día han desaparecido las limitaciones que la primera generación de PC presentaba con la limitación de memoria a 640 KBytes, no admitiéndose información a almacenar mayor de esa cantidad en el caso de computadoras IBM PC y compatibles).
- La transmisión de salida de resultados de un programa pueda ser tratada como entrada a otro programa.

[1] En sus orígenes y en la década de los ochenta, 640 K-bytes en el caso de las computadoras personales IBM PC y compatibles. Hoy día esas cifras han sido superadas con creces, pero aunque las memorias centrales varían, en computadoras domésticas, portátiles (*laptops*) y de escritorio, entre 1 GB y 4 GB, la temporalidad de los datos almacenados en ellas aconseja siempre el uso de archivos para datos de carácter permanente.

Para poder superar estas dificultades se necesitan dispositivos de almacenamiento secundario (memorias externas o auxiliares) como cintas, discos magnéticos, tarjetas perforadas, etc., donde se almacenará la información o datos que podrá ser recuperada para su tratamiento posterior. Las estructuras de datos aplicadas a colección de datos en almacenamientos secundarios se llaman *organización de archivos*. La noción de ***archivo o fichero*** está relacionada con los conceptos de:

- Almacenamiento permanente de datos.
- Fraccionamiento o partición de grandes volúmenes de información en unidades más pequeñas que puedan ser almacenadas en memoria central y procesadas por un programa.

Un *archivo* o *fichero* es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas *registros* o *artículos*, que son de igual tipo y constan a su vez de diferentes entidades de nivel más bajo denominadas *campos*.

9.1.1. Campos

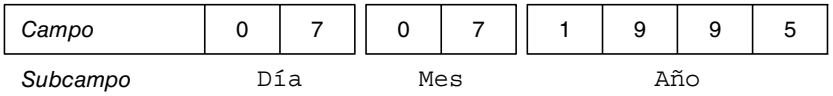
Un *campo* es un *item* o *elemento de datos elementales*, tales como un nombre, número de empleados, ciudad, número de identificación, etc.

Un campo está caracterizado por su tamaño o longitud y su tipo de datos (cadena de caracteres, entero, lógico, etcétera.). Los campos pueden incluso variar en longitud. En la mayoría de los lenguajes de programación los campos de longitud variable no están soportados y se suponen de longitud fija.

Figura 9.1. Campos de un registro.

Un campo es la unidad mínima de información de un registro.

Los datos contenidos en un campo se dividen con frecuencia en *subcampos*; por ejemplo, el campo fecha se divide en los subcampos día, mes, año.



Los rangos numéricos de variación de los subcampos anteriores son:

```
1 ≤ día ≤ 31
1 ≤ mes ≤ 12
1 ≤ año ≤ 1987
```

9.1.2. Registros

Un *registro* es una colección de información, normalmente relativa a una entidad particular. Un registro es una colección de campos lógicamente relacionados, que pueden ser tratados como una unidad por algún programa. Un ejemplo de un registro puede ser la información de un determinado empleado que contiene los campos de nombre, dirección, fecha de nacimiento, estudios, salario, trienios, etc.

Los registros pueden ser todos de *longitud fija*; por ejemplo, los registros de empleados pueden contener el mismo número de campos, cada uno de la misma longitud para nombre, dirección, fecha, etc. También pueden ser de *longitud variables*.

Los registros organizados en campos se denominan *registros lógicos*.

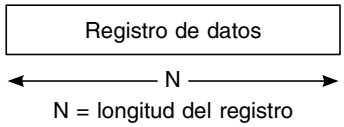


Figura 9.2. Registro.

Nota

El concepto de registro es similar al concepto de estructura (`struct`) estudiado en el Capítulo 7, ya que ambas estructuras de datos permiten almacenar datos de tipo heterogéneo.

9.1.3. Archivos (ficheros)

Un *fichero* (*archivo*) de datos —o simplemente un **archivo**— es una colección de registros relacionados entre sí con aspectos en común y organizados para un propósito específico. Por ejemplo, un fichero de una clase escolar contiene un conjunto de registros de los estudiantes de esa clase. Otros ejemplos pueden ser el fichero de nóminas de una empresa, inventarios, stocks, etc.

La Figura 9.3 recoge la estructura de un archivo correspondiente a los suscriptores de una revista de informática.

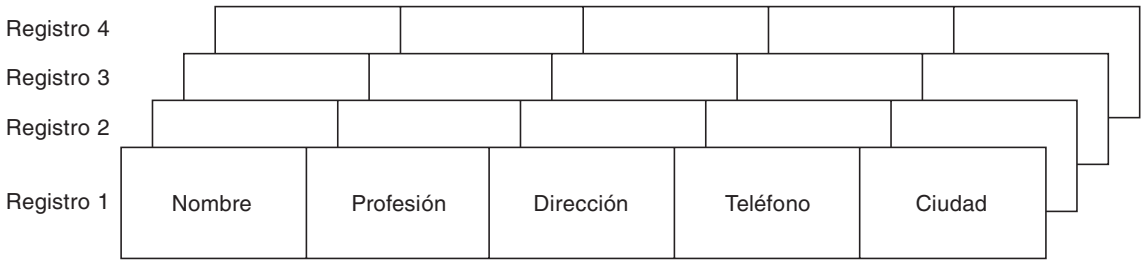


Figura 9.3. Estructuras de un archivo “suscriptores”.

Un archivo en una computadora es una estructura diseñada para contener datos. Los datos están organizados de tal modo que puedan ser recuperados fácilmente, actualizados o borrados y almacenados de nuevo en el archivo con todos los campos realizados.

9.1.4. Bases de datos

Una colección de archivos a los que puede accederse por un conjunto de programas y que contienen todos ellos datos relacionados constituye una base de datos. Así, una base de datos de una universidad puede contener archivos de estudiantes, archivos de nóminas, inventarios de equipos, etc.

9.1.5. Estructura jerárquica

Los conceptos carácter, campos, registro, archivo y base de datos son *conceptos lógicos* que se refieren al medio en que el usuario de computadoras ve los datos y se organizan. Las estructuras de datos se organizan de un modo jerárquico, de modo que el nivel más alto lo constituye la base de datos y el nivel más bajo el carácter.

9.1.6. Jerarquía de datos

Una computadora, como ya conoce el lector (Capítulo 1), procesa todos los datos como combinaciones de ceros y unos. Tal elemento de los datos se denomina bit (binary digit). Sin embargo, como se puede deducir fácilmente, es difícil para los programadores trabajar con datos en estos formatos de bits de bajo nivel. En su lugar, los programadores prefieren trabajar con caracteres tales como los dígitos decimales (`0-9`), letras (`A-Z` y `a-z`) o símbolos especiales (`&`, `*`, `,` `@`, €, #,...). El conjunto de todos los caracteres utilizados para escribir los programas se denomina *conjunto* o *juegos de caracteres* de la computadora. Cada carácter se representa como un patrón de ceros y unos. Por ejemplo, en Java, los caracteres son caracteres Unicode (Capítulo 1) compuestos de 2 bytes.

Al igual que los caracteres se componen de bits, los *campos* se componen de caracteres o bytes. Un **campo** es un grupo de caracteres o bytes que representan un significado. Por ejemplo, un campo puede constar de letras mayúsculas y minúsculas que representan el nombre de una ciudad.

Los datos procesados por las computadoras se organizan en *jerarquías de datos* formando estructuras a partir de bits, caracteres, campos, etc.

Los campos (variables de instancias en C++ y Java) se agrupan en *registros* que se implementan en una **clase** en Java o en C++. Un registro es un grupo de campos relacionados que se implementan con tipos de datos básicos o estructurados. En un sistema de matrícula en una universidad, un registro de un alumno o de un profesor puede constar de los siguientes campos:

- Nombre `(cadena)`.
- Número de expediente `(entero)`.
- Número de Documento Nacional de Identidad o Pasaporte `(entero doble)`.
- Año de nacimiento `(entero)`.
- Estudios `(cadena)`.

Un archivo es un grupo de registros relacionados. Así, una universidad puede tener muchos alumnos y profesores, y un archivo de alumnos contiene un registro para cada empleado. Un archivo de una universidad puede contener miles de registros y millones o incluso miles de millones de caracteres de información. Las Figura 9.4 muestra la *jerarquía de datos* de un archivo (*byte, campo, registro, archivo*).

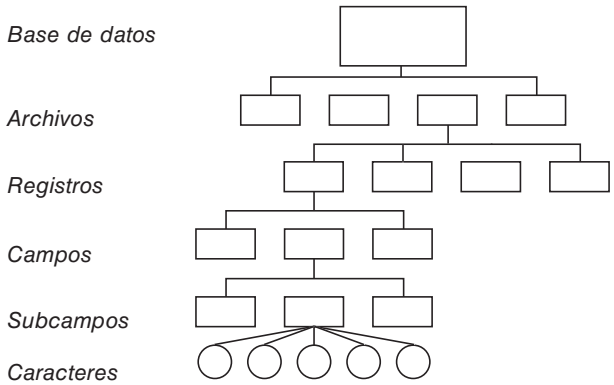


Figura 9.4. Estructuras jerárquicas de datos.

Los registros poseen una *clave* o llave que identifica a cada registro y que es única para diferenciarla de otros registros. En registros de nombres es usual que el campo clave sea el pasaporte o el DNI (Documento Nacional de Identidad).

Un conjunto de archivos relacionados se denomina base de datos. En los negocios o en la administración, los datos se almacenan en bases de datos y en muchos archivos diferentes. Por ejemplo, las universidades pueden tener archivos de profesores, archivos de estudiantes, archivos de planes de estudio, archivos de nóminas de profesores y de PAS (Personal de Administración y Servicios). Otra jerarquía de datos son los sistemas de gestión de bases de datos (**SGBD** o **DBMS**) que es un conjunto de programas diseñados para crear y administrar bases de datos.

9.2. CONCEPTOS Y DEFINICIONES = TERMINOLOGÍA

Aunque en el apartado anterior ya se han comentado algunos términos relativos a la teoría de archivos, en este apartado se enunciarán todos los términos más utilizados en la gestión y diseño de archivos.

9.2.1. Clave (indicativo)

Una *clave* (*key*) o *indicativo* es un campo de datos que identifica el registro y lo diferencia de otros registros. Esta clave debe ser diferente para cada registro. Claves típicas son nombres o números de identificación.

9.2.2. Registro físico o bloque

Un *registro físico* o *bloque* es la cantidad más pequeña de datos que pueden transferirse en una operación de entrada/salida entre la memoria central y los dispositivos periféricos o viceversa. Ejemplos de registros físicos son: una tarjeta perforada, una línea de impresión, un sector de un disco magnético, etc.

Un bloque puede contener uno o más registros lógicos.

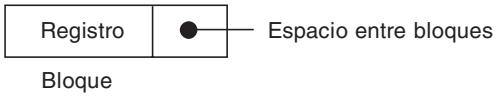
Un registro lógico puede ocupar menos de un registro físico, un registro físico o más de un registro físico.

9.2.3. Factor de bloqueo

Otra característica que es importante en relación con los archivos es el concepto de *factor de bloqueo* o *blocaje*. El número de registros lógicos que puede contener un registro físico se denomina factor de bloqueo.

Se pueden dar las siguientes situaciones:

- *Registro lógico* > *Registro físico.* En un bloque se contienen varios registros físicos por bloque; se denominan *registros expandidos.*
- *Registro lógico* = *Registro físico.* El factor de bloqueo es 1 y se dice que los registros *no están bloqueados.*
- *Registro lógico* < *Registro físico.* El factor de bloqueo es mayor que 1 y los registros *están bloqueados.*



a) Un registro por bloque (factor = 1)

b) N registros por bloque (factor = N)

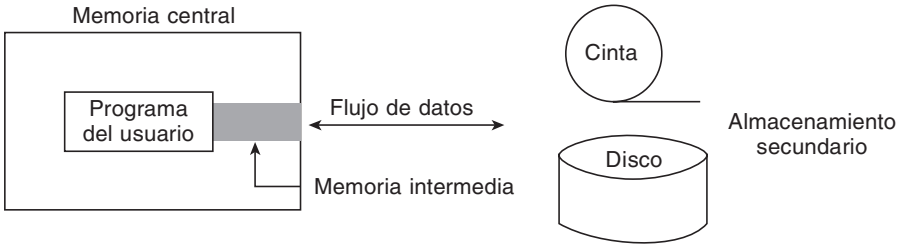
Figura 9.5. Factor de bloqueo.

La importancia del factor de bloqueo se puede apreciar mejor con un ejemplo. Supongamos que se tienen dos archivos. Uno de ellos tiene un factor de bloqueo de 1 (un registro en cada bloque). El otro archivo tiene un factor de bloqueo de 10 (10 registros/bloque). Si cada archivo contiene un millón de registros, el segundo archivo requerirá 900.000 operaciones de entrada/salida menos para leer todos los registros. En el caso de las computadoras personales con un tiempo medio de acceso de 90 milisegundos, el primer archivo emplearía alrededor de 24 horas más para leer todos los registros del archivo.

Un factor de bloqueo mayor que 1 siempre mejora el rendimiento; entonces, ¿por qué no incluir todos los registros en un solo bloque? La razón reside en que las operaciones de entrada/salida que se realizan por bloques se hacen a través de un área de la memoria central denominada *memoria intermedia (buffer)* y entonces el aumento del bloque implicará aumento de la memoria intermedia y, por consiguiente, se reducirá el tamaño de la memoria central.

El tamaño de una memoria intermedia de un archivo es el mismo que el del tamaño de un bloque. Como la memoria central es más cara que la memoria secundaria, no conviene aumentar el tamaño del bloque alegremente, sino más bien conseguir un equilibrio entre ambos criterios.

En el caso de las computadoras personales, el registro físico puede ser un sector del disco (512 bytes).



La Tabla 9.1 resume los conceptos lógicos y físicos de un registro.

Tabla 9.1. Unidades de datos lógicos y físicos

Organización lógica	Organización fiscia	Descripción
	Bit	Un dígito binario.
Carácter	Byte (octeto, 8 bits)	En la mayoría de los códigos un carácter se representa aproximadamente por un byte.
Campo	Palabra	Un campo es un conjunto relacionado de caracteres. Una palabra de computadora es un número fijo de bytes.
Registro	Bloque (1 página = bloques de longitud)	Los registros pueden estar bloqueados.
Archivo	Área	Varios archivos se pueden almacenar en un área de almacenamiento.
Base de datos	Áreas	Colección de archivos de datos relacionados que se pueden organizar en una base de datos.

Resumen de archivos

- Un archivo está siempre almacenado en un soporte externo a la memoria central.
- Existe independencia de las informaciones respecto de los programas.
- Todo programa de tratamiento intercambia información con el archivo y la unidad básica de entrada/salida es el registro.
- La información almacenada es permanente.
- En un momento dado, los datos extraídos por el archivo son los de un registro y no los del archivo completo.
- Los archivos en memoria auxiliar permiten una gran capacidad de almacenamiento.

9.3. SOPORTES SECUENCIALES Y DIRECCIONABLES

El soporte es el medio físico donde se almacenan los datos. Los tipos de soporte utilizados en la gestión de archivos son:

- *Soportes secuenciales.*
- *Soportes direccionables.*

Los *soportes secuenciales* son aquellos en los que los registros —informaciones— están escritos unos a continuación de otros y para acceder a un determinado registro *n* se necesita pasar por los *n – 1* registros anteriores.

Los *soportes direccionables* se estructuran de modo que las informaciones registradas se pueden localizar directamente por su dirección y no se requiere pasar por los registros precedentes. En estos soportes los registros deben poseer un campo clave que los diferencie del resto de los registros del archivo. Una dirección en un soporte direccionable puede ser número de pista y número de sector en un disco.

Los soportes direccionables son los discos magnéticos, aunque pueden actuar como soporte secuencial.

9.4. ORGANIZACIÓN DE ARCHIVOS

Según las características del soporte empleado y el modo en que se han organizado los registros, se consideran dos tipos de acceso a los registros de un archivo:

- *Acceso secuencial.*
- *Acceso directo.*

El *acceso secuencial* implica el acceso a un archivo según el orden de almacenamiento de sus registros, uno tras otro.

El *acceso directo* implica el acceso a un registro determinado, sin que ello implique la consulta de los registros precedentes. Este tipo de acceso sólo es posible con soportes direccionables.

La *organización* de un archivo define la forma en la que los registros se disponen sobre el soporte de almacenamiento, o también se define la organización como la forma en que se estructuran los datos en un archivo. En general, se consideran tres organizaciones fundamentales:

- *Organización secuencial.*
- *Organización directa o aleatoria ("random").*
- *Organización secuencial indexada ("indexed").*

9.4.1. Organización secuencial

Un archivo con organización secuencial es una sucesión de registros almacenados consecutivamente sobre el soporte externo, de tal modo que para acceder a un registro *n* dado es obligatorio pasar por todos los *n – 1* artículos que le preceden.

Los registros se graban consecutivamente cuando el archivo se crea y se debe acceder consecutivamente cuando se leen dichos registros.

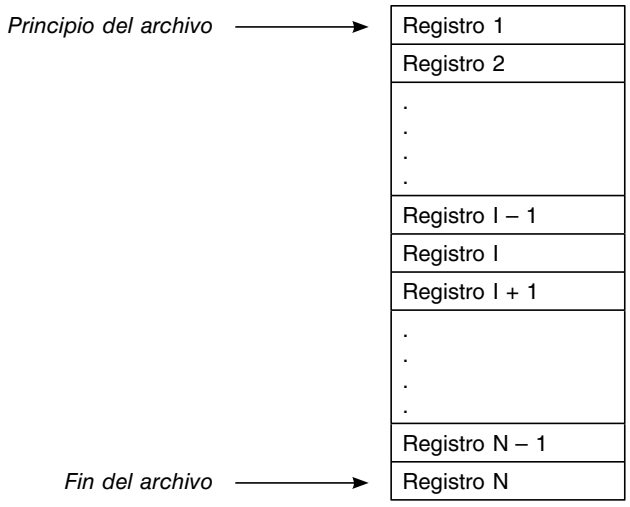


Figura 9.6. Organización secuencial.

- El orden físico en que fueron grabados (escritos) los registros es el orden de lectura de los mismos.
- Todos los tipos de dispositivos de memoria auxiliar soportan la organización secuencial.

Los archivos organizados secuencialmente contienen un registro particular —el último— que contiene una marca fin de archivo (`EOF` o bien `FF`). Esta marca fin de archivo puede ser un carácter especial como `'*'`.

9.4.2. Organización directa

Un archivo está organizado en modo directo cuando el orden físico no se corresponde con el orden lógico. Los datos se sitúan en el archivo y se accede a ellos directamente mediante su posición, es decir, el lugar relativo que ocupan.

Esta organización tiene la *ventaja* de que se pueden leer y escribir registros en cualquier orden y posición. Son muy rápidos de acceso a la información que contienen.

La organización directa tiene el *inconveniente* de que necesita programar la relación existente entre el contenido de un registro y la posición que ocupa. El acceso a los registros en modo directo implica la posible existencia de huecos libres dentro del soporte y, por consecuencia, pueden existir huecos libres entre registros.

La correspondencia entre clave y dirección debe poder ser programada y la determinación de la relación entre el registro y su posición física se obtiene mediante una fórmula.

Las condiciones para que un archivo sea de organización directa son:

- Almacenado en un soporte direccionable.
- Los registros deben contener un campo específico denominado *clave* que identifica cada registro de modo único, es decir, dos registros distintos no pueden tener un mismo valor de clave.
- Existencia de una correspondencia entre los posibles valores de la clave y las direcciones disponibles sobre el soporte.

Un soporte direccionable es normalmente un disco o paquete de discos. Cada posición se localiza por su *dirección absoluta*, que en el caso del disco suele venir definida por dos parámetros —número de pista y número de sector— o bien por tres parámetros —pista, sector y número de cilindro—; un *cilindro i* es el conjunto de pistas de número *i* de cada superficie de almacenamiento de la pila.

En la práctica el programador no gestiona directamente direcciones absolutas, sino *direcciones relativas* respecto al principio del archivo. La manipulación de direcciones relativas permite diseñar el programa con independencia de la posición absoluta del archivo en el soporte.

El programador crea una relación perfectamente definida entre la clave indicativa de cada registro y su posición física dentro del dispositivo de almacenamiento. Esta relación, en ocasiones, produce *colisiones.*

Consideremos a continuación el fenómeno de las *colisiones* mediante un ejemplo.

La clave de los registros de estudiantes de una Facultad de Ciencias es el número de expediente escolar que se le asigna en el momento de la matriculación y que consta de ocho dígitos. Si el número de estudiantes es un número decimal de ocho dígitos, existen 10^8 posibles números de estudiantes (0 a 99999999), aunque lógicamente *nunca* existirán tantos estudiantes (incluso incluyendo alumnos ya graduados). El archivo de estudiantes constará a lo sumo de decenas o centenas de miles de estudiantes. Se desea almacenar este archivo en un disco sin utilizar mucho espacio. Si se desea obtener el algoritmo de direccionamiento, se necesita una *función de conversión de claves o función "hash"*. Suponiendo que N es el número de posiciones disponibles para el archivo, el algoritmo de direccionamiento convierte cada valor de la clave en una dirección relativa *d*, comprendida entre 1 y *N*. Como la clave puede ser numérica o alfanumérica, el algoritmo de conversión debe prever esta posibilidad y asignar a cada registro correspondiente a una clave una posición física en el soporte de almacenamiento. Así mismo, el algoritmo o función de conversión de claves debe eliminar o reducir al máximo las colisiones. Se dice que en un algoritmo de conversión de claves se produce una *colisión* cuando dos registros de claves distintas producen la misma dirección física en el soporte. El *inconveniente de una colisión* radica en el hecho de tener que situar el registro en una posición diferente de la indicada por el algoritmo de conversión y, por consiguiente, el acceso a este registro será más lento. Las colisiones son difíciles de evitar en las organizaciones directas. Sin embargo, un tratamiento adecuado en las operaciones de lectura/escritura disminuirá su efecto perjudicial en el archivo.

Para representar la función de transformación o conversión de claves (*hash*), se puede utilizar una notación matemática. Así, si K es una clave, f(K) es la correspondiente dirección; f es la función llamada *función de conversión*.

EJEMPLO 9.1

Una compañía de empleados tiene un número determinado de vendedores y un archivo en el que cada registro corresponde a un vendedor. Existen 200 vendedores, cada uno referenciado por un número de cinco dígitos. Si tuviésemos que asignar un archivo de 100.000 registros, cada registro se corresponderá con una posición del disco.

Para el diseño del archivo crearemos 250 registros (un 25 por 100 más que el número de registros necesarios —25 por 100 suele ser un porcentaje habitual—) que se distribuirán de la siguiente forma:

1. Posiciones 0-199 constituyen el área principal del archivo y en ella se almacenarán todos los vendedores.
2. Posiciones 200-249 constituyen el área de desbordamiento, si $K(1) <> K(2)$, pero $f(K(1)) = f(K(2))$, y el registro con clave $K(1)$ ya está almacenado en el área principal, entonces el registro con $K(2)$ se almacena en el área de desbordamiento.

La función *f* se puede definir como:

$f(k)$ = resto cuando K se divide por 199, esto es, el módulo de 199; 199 ha sido elegido por ser el número primo mayor y que es menor que el tamaño del área principal.

Para establecer el archivo se borran primero 250 posiciones. A continuación, para cada registro de vendedor se calcula $p = f(K)$. Si la posición p está vacía, se almacena el registro en ella. En caso contrario se busca secuencialmente a través de las posiciones 200, 201, ..., para el registro con la clave deseada.

9.4.3. Organización secuencial indexada

Un diccionario es un archivo secuencial, cuyos registros son las entradas y cuyas claves son las palabras definidas por las entradas. Para buscar una palabra (una clave) no se busca secuencialmente desde la "a" hasta la "z", sino que se abre el diccionario por la letra inicial de la palabra. Si se desea buscar "índice", se abre el índice por la letra *I* y en su primera página se busca la cabecera de página hasta encontrar la página más próxima a la palabra, buscando a continuación palabra a palabra hasta encontrar "índice". El diccionario es un ejemplo típico de archivo secuencial indexado con dos niveles de índices, el nivel superior para las letras iniciales y el nivel menor para las cabeceras de página. En una organización de computadora las letras y las cabeceras de páginas se guardarán en un archivo de índice independiente de las entradas del diccionario (archivo de datos). Por consiguiente, cada archivo secuencial indexado consta de un archivo índice y un archivo de datos.

Un archivo está organizado en forma secuencial indexada si:

- El tipo de sus registros contiene un campo clave identificador.
- Los registros están situados en un soporte direccionable por el orden de los valores indicados por la clave.
- Un índice para cada posición direccionable, la dirección de la posición y el valor de la clave; en esencia, el índice contiene la clave del último registro y la dirección de acceso al primer registro del bloque.

Un archivo en organización secuencial indexada consta de las siguientes partes:

- *Área de datos o primaria*: contiene los registros en forma secuencial y está organizada en secuencia de claves sin dejar huecos intercalados.
- *Área de índices*: es una tabla que contiene los niveles de índice, la existencia de varios índices enlazados se denomina *nivel de indexación*.
- *Área de desbordamiento o excedentes*: utilizada, si fuese necesario, para las actualizaciones.

El área de índices es equivalente, en su función, al índice de un libro. En ella se refleja el valor de la clave identificativa más alta de cada grupo de registros del archivo y la dirección de almacenamiento del grupo.

Los archivos secuenciales indexados presentan las siguientes *ventajas*:

- Rápido acceso.
- El sistema de gestión de archivos se encarga de relacionar la posición de cada registro con su contenido mediante la tabla de índices.

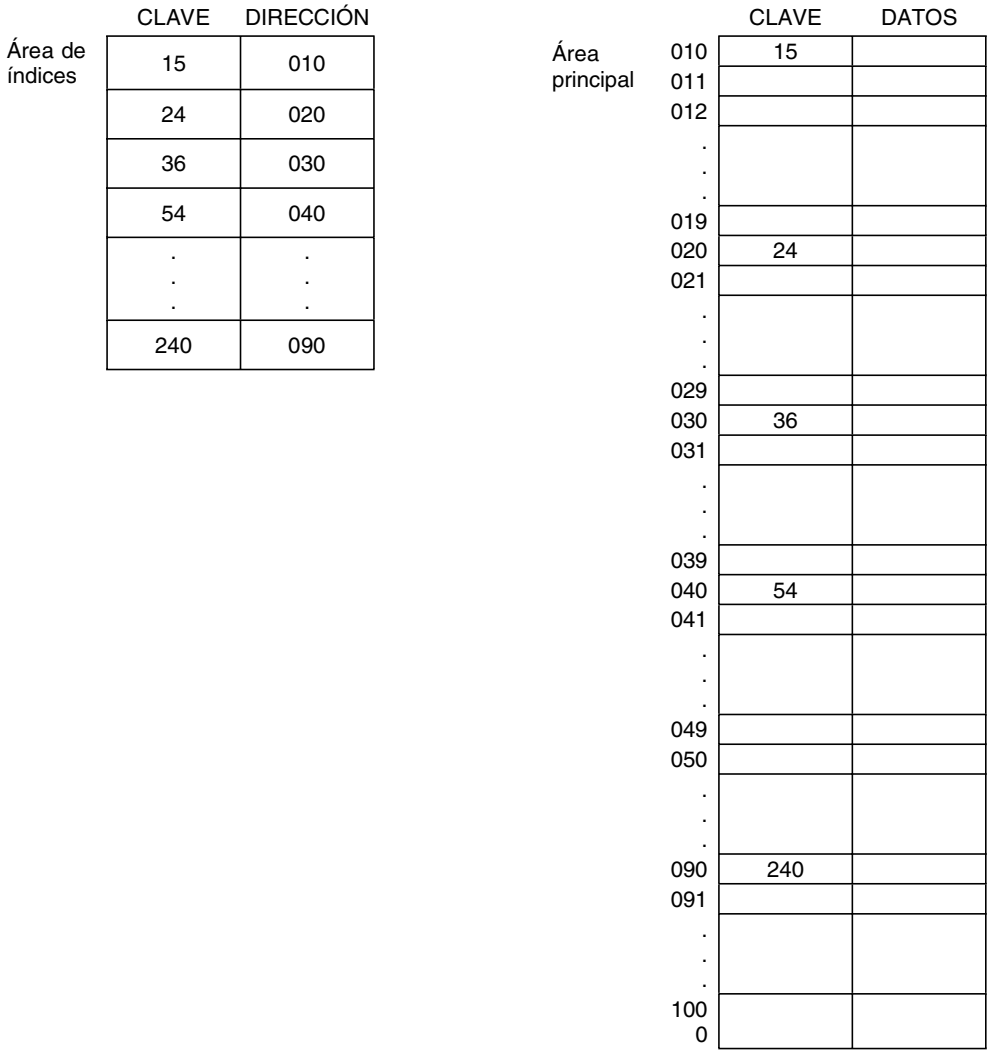


Figura 9.7. Organización secuencial indexada.

Y los siguientes *inconvenientes*:

- Desaprovechamiento del espacio por quedar huecos intermedios cada vez que se actualiza el archivo.
- Se necesita espacio adicional para el área de índices.

Los soportes que se utilizan para esta organización son los que permiten el acceso directo —los discos magnéticos—. Los soportes de acceso secuencial no pueden utilizarse, ya que no disponen de direcciones para las posiciones de almacenamiento.

9.5. OPERACIONES SOBRE ARCHIVOS

Tras la decisión del tipo de organización que ha de tener el archivo y los métodos de acceso que se van a aplicar para su manipulación, es preciso considerar todas las posibles operaciones que conciernen a los registros de un archivo. Las distintas operaciones que se pueden realizar son:

- *Creación.*
- *Consulta.*
- *Actualización* (altas, bajas, modificación, consulta).

- *Clasificación.*
- *Reorganización.*
- *Destrucción* (borrado).
- *Reunión, fusión.*
- *Rotura, estallido.*

9.5.1. Creación de un archivo

Es la primera operación que sufrirá el archivo de datos. Implica la elección de un entorno descriptivo que permita un ágil, rápido y eficaz tratamiento del archivo.

Para utilizar un archivo, éste tiene que existir, es decir, las informaciones de este archivo tienen que haber sido almacenadas sobre un soporte y ser utilizables. La *creación* exige organización, estructura, localización o reserva de espacio en el soporte de almacenamiento, transferencia del archivo del soporte antiguo al nuevo.

Un archivo puede ser creado por primera vez en un soporte, proceder de otro previamente existente en el mismo o diferente soporte, ser el resultado de un cálculo o ambas cosas a la vez.

La Figura 9.8 muestra un organigrama de la creación de un archivo ordenado de empleados de una empresa por el campo clave (número o código de empleado).

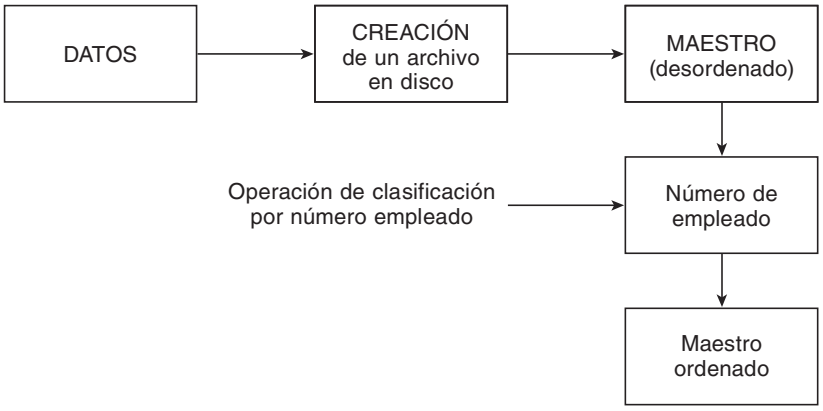


Figura 9.8. Creación de un archivo ordenado de empleados.

9.5.2. Consulta de un archivo

Es la operación que permite al usuario acceder al archivo de datos para conocer el contenido de uno, varios o todos los registros.

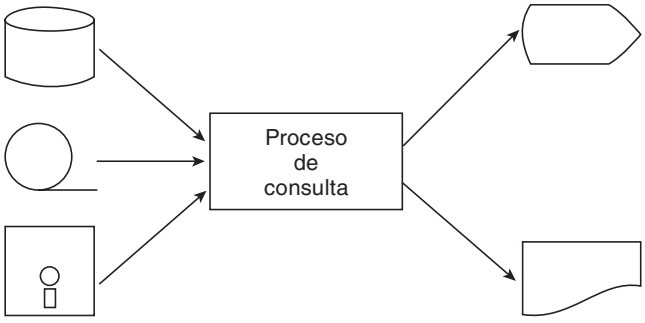


Figura 9.9. Consulta de un archivo.

9.5.3. Actualización de un archivo

Es la operación que permite tener actualizado (puesto al día) el archivo, de tal modo que sea posible realizar las siguientes operaciones con sus registros:

- *Consulta* del contenido de un registro.
- *Inserción* de un registro nuevo en el archivo.
- *Supresión* de un registro existente.
- *Modificación* de un registro.

Un ejemplo de actualización es el de un archivo de un almacén, cuyos registros contienen las existencias de cada artículo, precios, proveedores, etc. Las existencias, precios, etc., varían continuamente y exigen una actualización simultánea del archivo con cada operación de consulta.

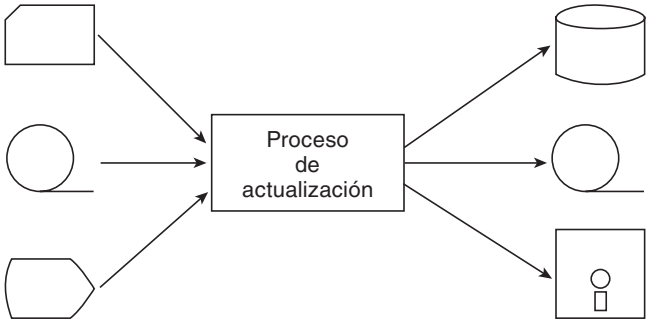


Figura 9.10. Actualización de un archivo (I).

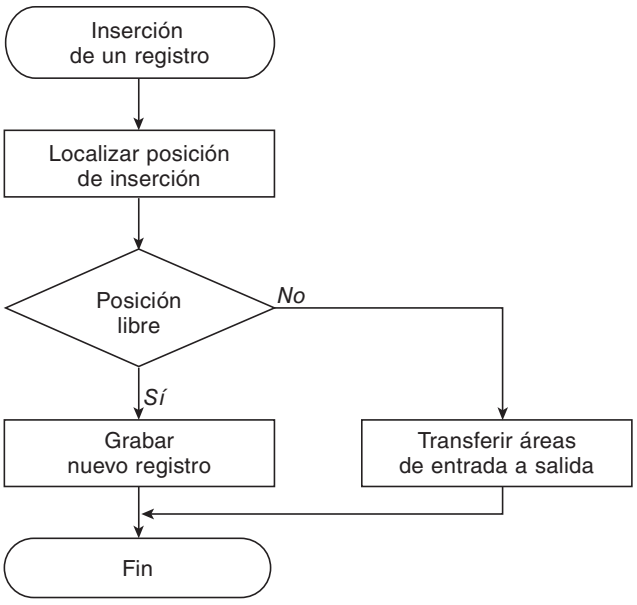


Figura 9.11. Actualización de un archivo (II).

9.5.4. Clasificación de un archivo

Una operación muy importante en un archivo es la *clasificación u ordenación* (*sort*, en inglés). Esta clasificación se realizará de acuerdo con el valor de un campo específico, pudiendo ser *ascendente* (creciente) o *descendente* (decreciente): alfabética o numérica (véase Figura 9.12).

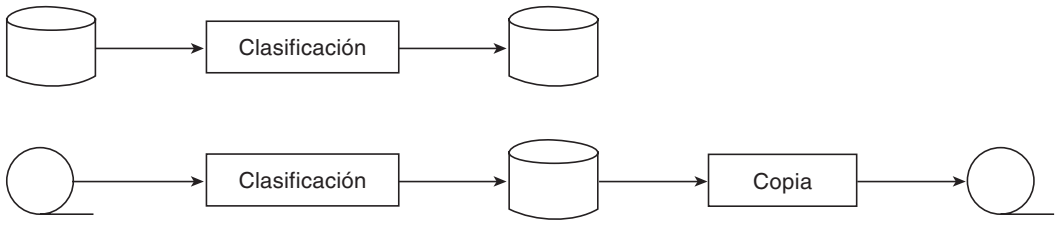


Figura 9.12. Clasificación de un archivo.

9.5.5. Reorganización de un archivo

Las operaciones sobre archivos modifican la estructura inicial o la óptima de un archivo. Los índices, enlaces (punteros), zonas de sinónimos, zonas de desbordamiento, etc., se modifican con el paso del tiempo, lo que hace a la operación de acceso al registro cada vez más lenta.

La reorganización suele consistir en la copia de un nuevo archivo a partir del archivo modificado, a fin de obtener una nueva estructura lo más óptima posible.

9.5.6. Destrucción de un archivo

Es la operación inversa a la creación de un archivo (*kill*, en inglés). Cuando se destruye (anula o borra) un archivo, éste ya no se puede utilizar y, por consiguiente, no se podrá acceder a ninguno de sus registros (Figura 9.13).

9.5.7. Reunión, fusión de un archivo

Reunión. Esta operación permite obtener un archivo a partir de otros varios (Figura 9.14).

Fusión. Se realiza una fusión cuando se reúnen varios archivos en uno solo, intercalándose unos en otros, siguiendo unos criterios determinados.

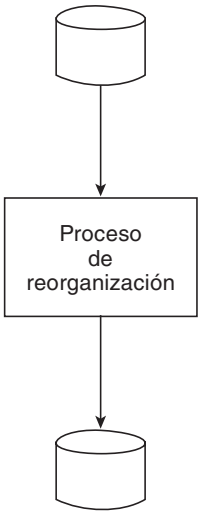


Figura 9.13. Reorganización de un archivo.

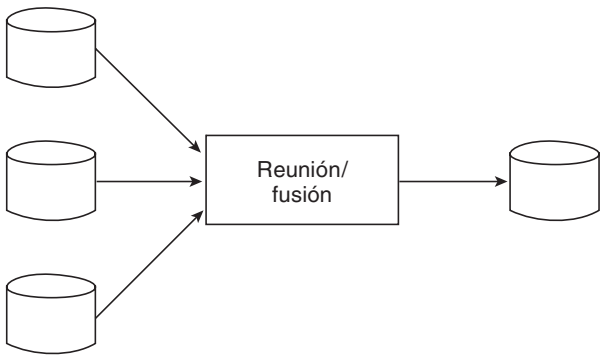


Figura 9.14. Fusión de archivos.

9.5.8. Rotura/estallido de un archivo

Es la operación de obtener varios archivos a partir de un mismo archivo inicial.

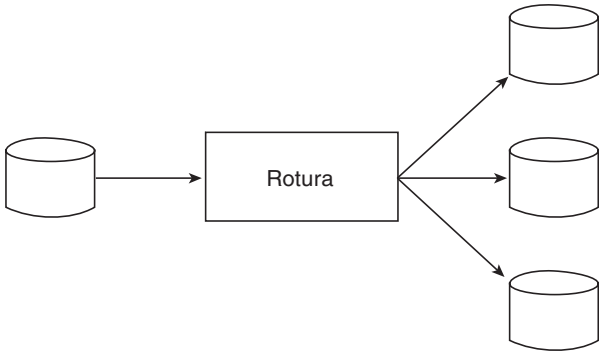


Figura 9.15. Rotura de un archivo.

9.6. GESTIÓN DE ARCHIVOS

Las operaciones sobre archivos se realizan mediante programas y el primer paso para poder gestionar un archivo mediante un programa es declarar un identificador lógico que se asocie al nombre externo del archivo para permitir su manipulación. La declaración se realizará con una serie de instrucciones como las que se muestran a continuación, cuya asociación permite establecer la organización del archivo y estructura de sus registros lógicos.

```
tipo
 registro: <tipo_registro>
   <tipo>:<nombre del campo>
    ....
  fin_registro
archivo_<organización> de <tipo_de_dato>:<tipo_archivo

var
 tipo_registro: nombre_registro
 tipo_archivo:identificador_archivo

tipo
  registro: Rempleado
    cadena: nombre
    cadena: cod
    entero: edad
    real: salario
  fin_registro
archivo_d de rempleado:empleado
var
  Rempleado: Re
  Empleado: E
```

Las operaciones, básicas para la gestión de archivos, que *tratan con la propia estructura del archivo* se consideran predefinidas y son:

- *Crear archivos (create).* Consiste en definirlo mediante un nombre y unos atributos. Si el archivo existiera con anterioridad lo destruiría.

- *Abrir o arrancar (open)* un archivo que fue creado con anterioridad a la ejecución de este programa. Esta operación establece la comunicación de la CPU con el soporte físico del archivo, de forma que los registros se vuelven accesibles para lectura, escritura o lectura/escritura.
- *Incrementar* o *ampliar* el tamaño del archivo (*append*, *extend*).
- *Cerrar* el archivo después que el programa ha terminado de utilizarlo *(close).* Cierra la comunicación entre la CPU y el soporte físico del archivo.
- *Borrar (delete)* un archivo que ya existe. Borra el archivo del soporte físico, liberando espacio.
- *Transferir datos desde (leer) o a (escribir)* el dispositivo diseñado por el programa. Estas operaciones copian los registros del archivo sobre variables en memoria central y viceversa.

En cuanto a las operaciones más usuales en los registros son:

- *Consulta*: lectura del contenido de un registro.
- *Modificación*: alterar la información contenida en un registro.
- *Inserción*: añadir un nuevo registro al archivo.
- *Borrado*: suprimir un registro del archivo.

9.6.1. Crear un archivo

La creación de un archivo es la operación mediante la cual se introduce la información correspondiente al archivo en un soporte de almacenamiento de datos.

Antes de que cualquier usuario pueda procesar un archivo es preciso que éste haya sido creado previamente. El proceso de creación de un archivo será la primera operación a realizar. Una vez que el archivo ha sido creado, la mayoría de los usuarios simplemente desearán acceder al archivo y a la información contenida en él.

Para crear un nuevo archivo dentro de un sistema de computadora se necesitan los siguientes datos:

- *Nombre dispositivo*: indica el lugar donde se situará el archivo cuando se cree.
- *Nombre del archivo*: identifica el archivo entre los restantes archivos de una computadora.
- *Tamaño del archivo*: indica el espacio necesario para la creación del archivo.
- *Organización del archivo*: tipo de organización del archivo.
- *Tamaño del bloque o registro físico*: cantidad de datos que se leen o escriben en cada operación de entrada/salida (E/S).

Al ejecutar la creación de un archivo se pueden generar una *serie de errores,* entre los que se pueden destacar los siguientes:

- Otro archivo con el mismo nombre ya existía en el soporte.
- El dispositivo no tiene espacio disponible para crear otro nuevo archivo.
- El dispositivo no está operacional.
- Existe un problema de hardware que hace abortar el proceso.
- Uno o más de los parámetros de entrada en la instrucción son erróneos.

La instrucción o acción en pseudocódigo que permite crear un archivo se codifica con la palabra **crear**.

```
crear(<var_tipo_archivo>, <nombre_físico>)
```

9.6.2. Abrir un archivo

La acción de *abrir (open)* un archivo es permitir al usuario localizar y acceder a los archivos que fueron creados anteriormente.

La diferencia esencial entre una instrucción de *abrir* un archivo y una instrucción de *crear* un archivo residen en que el archivo no existe antes de utilizar **crear** y se supone que debe existir antes de utilizar **abrir**.

La información que un sistema de tratamiento de archivos requiere para abrir un archivo es diferente de las listas de información requerida para crear un archivo. La razón para ello reside en el hecho que toda la información que realmente describe el archivo se escribió en éste durante el proceso de creación del archivo. Por consiguiente, la operación **`crear`** sólo necesita localizar y leer esta información conocida como atributos del archivo.

La instrucción de abrir un archivo consiste en la creación de un canal que comunica a un usuario a través de un programa con el archivo correspondiente situado en un soporte.

Los parámetros que se deben incluir en una instrucción de apertura (**`abrir`**) son:

- Nombre del dispositivo.
- Nombre del usuario o canal de comunicación.
- Nombre del archivo.

Al ejecutar la instrucción **`abrir`** se pueden encontrar los siguientes errores:

- Archivo no encontrado en el dispositivo especificado (nombre de archivo o identificador de dispositivo erróneo).
- Archivo ya está en uso para alguna otra aplicación del usuario.
- Errores hardware.

El formato de la instrucción es:

```
Abrir (<var_tipo_archivo>,<modo>,<nombre_físico>)
```

La operación de abrir archivos se puede aplicar para operaciones de lectura (l), escritura (e), lectura/escritura (l/e).

```
abrir (id_archivo, l, nombre_archivo)
```

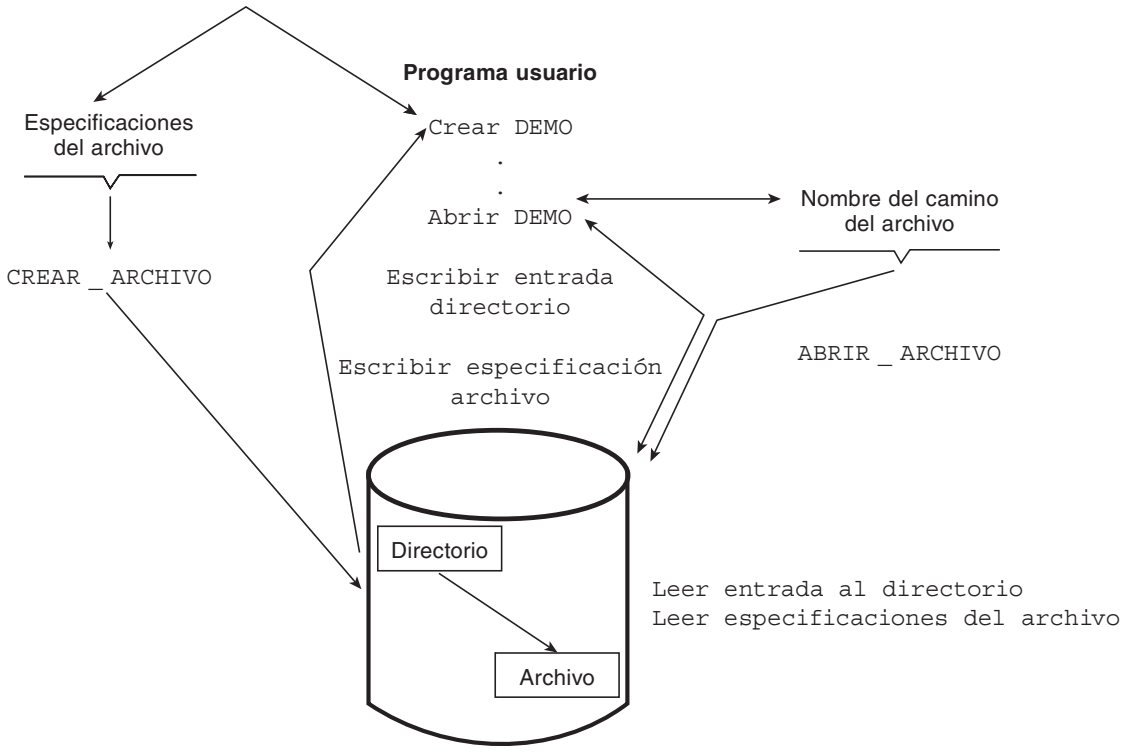


Figura 9.16. Abrir un archivo.

Para que un archivo pueda abrirse ha de haber sido previamente creado. Cuando un archivo se abre para lectura colocamos un hipotético puntero en el primer registro del archivo y se permitirán únicamente operaciones de lectura de los registros del archivo. La apertura para escritura coloca dicho hipotético puntero detrás del último registro del archivo, y dispuesto para la adición de nuevos registros en él. Ambos modos se consideran propios de archivos secuenciales. Los archivos directos se abrirán en modo lectura/escritura, permitiéndose tanto la lectura como la escritura de nuevos registros.

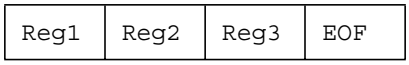
9.6.3. Cerrar archivos

El propósito de la operación de cerrar un archivo es permitir al usuario cortar el acceso o detener el uso del archivo, permitiendo a otros usuarios acceder al archivo. Para ejecutar esta función, el sistema de tratamiento de archivos sólo necesita conocer el nombre del archivo que se debe cerrar, y que previamente debía estar abierto.

Formato:

```
cerrar (<var_tipo-archivo>
```

Estructura:



9.6.4. Borrar archivos

La instrucción de **`borrar`** tiene como objetivo la supresión de un archivo del soporte o dispositivo. El espacio utilizado por un archivo borrado puede ser utilizado para otros archivos.

La información necesaria para eliminar un archivo es:

- Nombre del dispositivo y número del canal de comunicación.
- Nombre del archivo.

Los *errores* que se pueden producir son:

- El archivo no se puede encontrar bien porque el nombre no es válido o porque nunca existió.
- Otros usuarios estaban actuando sobre el archivo y estaba activo.
- Se detectó un problema de hardware.

9.7. FLUJOS

Un **archivo** o **fichero** es una colección de datos relacionados. En esencia, C++ o Java visualizan cada archivo como un **flujo** *(stream)* secuencial de bytes. En la entrada, un programa extrae bytes de un flujo de entrada y en la salida, un programa inserta bytes en el flujo de salida. En un programa orientado a texto, cada byte representa un carácter; en general, los bytes pueden formar una representación binaria de datos carácter o numéricos. Los bytes de un flujo de entrada pueden venir del teclado o de un escáner, por ejemplo, pero también pueden venir de un dispositivo de almacenamiento, tal como un disco duro o un CD, o desde otro programa. De modo similar, en un flujo de salida, los bytes pueden fluir a la pantalla, a una impresora, a un dispositivo de almacenamiento o a otro programa. En resumen, un flujo actúa como un intermediario entre el programa y el destino o fuente del flujo.

Este enfoque permite a un programa C++, Java,... tratar la entrada desde un archivo. En realidad el lenguaje trata un archivo como una serie de bytes; muchos archivos residen en un disco, pero dispositivos tales como impresoras, discos magnéticos y ópticos, y líneas de comunicación se consideran archivos. Con este enfoque, por ejemplo, un programa C++ examina el flujo de bytes sin necesidad de conocer su procedencia, y puede procesar la salida de modo independiente adonde vayan los bytes.

Un archivo es un flujo secuencial de bytes. Cada archivo termina con una marca final de archivo (EOF, end-of-file) o en un número de byte específico grabado en el sistema. Un programa que procesa un flujo de byte recibe una indicación del sistema cuando se alcanza el final del flujo con independencia de cómo estén representados los flujos o archivos.

9.7.1. Tipos de flujos

Existen dos tipos de flujos en función del sentido del canal de comunicación: flujo de entrada y flujo de salida. Un **flujo de entrada** lee información como una secuencia de caracteres. Estos caracteres pueden ser tecleados en la consola de entrada, leídos de un archivo de entrada, o leídos de zócalo de una red. Un **flujo de salida** es una secuencia de caracteres que se almacenan como información. Estos caracteres se pueden visualizar en la consola, escribir en un archivo de salida o en zócalos de red.

Un *flujo* de entrada envía datos desde una fuente a un programa. Un flujo de *salida* envía datos desde un programa a un destino.

Desde el punto de vista de la información que contienen, los flujos se clasifican en:

- *Flujos de bytes*, se utilizan para manejar bytes, enteros y otros tipos de datos simples. Un tipo muy diverso se pueden expresar en formato bytes, incluyendo datos numéricos, programas ejecutables, comunicaciones de Internet, *bytecode* (archivos de clases ejecutados por una máquina virtual Java). Cada tipo de dato se puede expresar o bien como bytes individuales o como combinación de bytes.
- *Flujos de caracteres*, manipulan archivos de texto y otras fuentes de texto. Se diferencian de los flujos de bytes en que soportan el conjunto de caracteres ASCII o Unicode. Cualquier tipo de datos que implique texto debe utilizar flujo de caracteres, incluyendo archivos de texto, páginas web o sitios comunes de texto.

9.7.2. Flujos en C++

La gestión de la entrada, implica dos etapas:

- Asociación de un flujo con una entrada a un programa.
- Conexión del flujo a un archivo.

En otras palabras, un flujo de entrada necesita dos conexiones, una en cada extremo. La conexión fin de archivo proporciona una fuente para el flujo y la conexión fin de programa vuelca el flujo de salida al programa (la conexión final de archivo, pero también puede ser un dispositivo, tal como un teclado). De igual modo, la gestión de salida implica la conexión de un flujo de salida al programa y la asociación de un destino de salida con el flujo. Al igual que sucede en una tubería del servicio del agua corriente de su ciudad, fluyen bytes en lugar de agua.

En C++ un flujo es un tipo especial de variable conocida como un objeto. Los flujos cin y cout se utilizan en entradas y salidas. La clase istream define el operador de extracción (>>) para los tipos primitivos. Este operador convierte los datos a una secuencia de caracteres y los inserta en el flujo.

Los flujos cin y cout se declaran en el lenguaje por usted, pero si desea que un flujo se conecte a un archivo, se debe declarar justo antes de que se pueda declarar cualquier otra variable.

9.7.3. Flujos en Java

El procedimiento para utilizar bien un flujo de bytes o un flujo de caracteres en Java es, en gran medida, el mismo. Antes de comenzar a trabajar con las clases específicas de la biblioteca de clases java.io, es útil revisar el proceso de crear y utilizar flujos.

Para un flujo de entrada, el primer paso es crear un objeto asociado con la fuente de datos. Por ejemplo, si la fuente es un archivo de su unidad de disco duro, un objeto FileInputStream se puede asociar con este archivo.

Después que se tiene un objeto de flujo, se puede leer la información desde el flujo utilizando uno de los métodos del objeto `FileInputStream` incluye un método `read` que devuelve un byte leído desde el teclado.

Cuando se termina de leer la información del flujo se llama al método `close( )` para indicar que se ha terminado de utilizar el flujo.

En el caso de un flujo de salida, se crea un objeto asociado con el destino de los datos. Tal objeto se puede crear de la clase `BufferedWriter` que representa un medio eficiente de crear archivos de texto.

El método `write( )` es el medio más simple para enviar información al destino del flujo de salida. Al igual que con los flujos de entrada, el método `close( )` se llama en un flujo de salida cuando no se tiene más información que enviar.

9.7.4. Consideraciones prácticas en Java y C#

Java y C# realizan las operaciones en archivos a través de flujos, manipulados por clases, que conectan con el medio de almacenamiento. De esta forma, para crear y abrir un archivo, se requiere utilizar una clase que defina la funcionalidad del flujo. Los flujos determinan el sentido de la comunicación (lectura, escritura, o lectura/escritura), la posibilidad de posicionamiento directo o no en un determinado registro y la forma de leer y/o escribir en el archivo. Cerrar el archivo implica cerrar el flujo. Así la siguiente instrucción en Java crea un flujo que permite la lectura/escritura (`rw`) en un archivo donde se podrá efectuar posicionamiento directo y cuyo nombre externo es `empleados.dat`.

```
RandomAccessFile e = new RandomAccessFile ("empleados.dat", "rw");
```

Pueden utilizarse flujos de bytes, caracteres, cadenas o tipos primitivos. Por ejemplo, en Java la clase `FileInputStream` permite crear un flujo para lectura secuencial de bytes desde un archivo, mientras `FileReader` lo crea para la lectura secuencial de caracteres y `RandomAccessFile`, como ya se ha comentado, admite posicionamiento directo y permite la lectura/escritura de datos tipos primitivos.

La personalización de flujos se consigue por asociación o encadenamiento de otros flujos sobre los flujos base de apertura de archivos. Una aplicación práctica de esta propiedad en Java puede ser permitir la lectura de una cadena de caracteres desde un flujo de entrada

```
BufferedReader f = new BufferedReader (new FileReader("datos.txt"));
cadena = f.readLine();      //lee una cadena del archivo
f.close();                  // cierra el archivo
```

En C# la situación es similar y sobre los flujos base, que conectan al medio de almacenamiento, pueden encadenarse otros para efectuar tratamientos especiales de la información.

```
BinaryWriter f = new BinaryWriter (new FileStream("notas.dat",
                 FileMode.OpenOrCreate, FileAccess.Write));
/* BinaryWriter proporciona métodos para escribir tipos de
 datos primitivos en formato binario */

f.Write (5.34 * 2);

f.Close();      // Cerrar el archivo
```

9.8. MANTENIMIENTO DE ARCHIVOS

La operación de mantenimiento de un archivo incluye todas las operaciones que sufre un archivo durante su vida y desde su creación hasta su eliminación o borrado.

El mantenimiento de un archivo consta de dos operaciones diferentes:

- *actualización,*
- *consulta.*

La *actualización* es la operación de eliminar o modificar los datos ya existentes, o bien introducir nuevos datos. En esencia, es la puesta al día de los datos del archivo.

Las operaciones de actualización son:

- *altas,*
- *bajas,*
- *modificaciones.*

Las operaciones de consulta tienen como finalidad obtener información total o parcial de los datos almacenados en un archivo y presentarlos en dispositivos de salida: pantalla o impresora, bien como resultados o como listados.

Todas las operaciones de mantenimiento de archivos suelen constituir módulos independientes del programa principal y su diseño se realiza con subprogramas (*subrutinas* o *procedimientos* específicos).

Así, los subprogramas de mantenimiento de un archivo constarán de:

Altas

Una operación de *alta* en un archivo consiste en la adición de un nuevo registro. En un archivo de empleados, un alta consistirá en introducir los datos de un nuevo empleado. Para situar correctamente un alta, se deberá conocer la posición donde se desea almacenar el registro correspondiente: al principio, en el interior o al final de un archivo.

El algoritmo del subprograma `ALTAS` debe contemplar la comprobación de que el registro a dar de alta no existe previamente.

Bajas

Una *baja* es la acción de eliminar un registro de un archivo. La baja de un registro se puede presentar de dos formas distintas: indicación del registro específico que se desea dar de baja o bien visualizar los registros del archivo para que el usuario elija el registro a borrar.

La baja de un registro puede ser *lógica* o *física*. Una *baja lógica* supone el no borrado del registro en el archivo. Esta baja lógica se manifiesta en un determinado campo del registro con una *bandera, indicador* o *"flag"* —carácter *, $, etc.—, o bien con la escritura o rellenado con espacios en blanco de algún campo en el registro específico.

Una *baja física* implica el borrado y desaparición del registro, de modo que se crea un nuevo archivo que no incluye el registro dado de baja.

Modificaciones

Una *modificación* en un archivo consiste en la operación de cambiar total o parcialmente el contenido de uno de sus registros.

Esta fase es típica cuando cambia el contenido de un determinado campo de un archivo; por ejemplo, la dirección o la edad de un empleado.

La forma práctica de modificar un registro es la visualización del contenido de sus campos; para ello se debe elegir el registro o registros a modificar. El proceso consiste en la lectura del registro, modificación de su contenido y escritura, total o parcial del mismo.

Consulta

La operación de *consulta* tiene como fin visualizar la información contenida en el archivo, bien de un modo completo —bien de modo parcial—, examen de uno o más registros.

Las operaciones de consulta de archivo deben contemplar diversos aspectos que faciliten la posibilidad de conservación de datos. Los aspectos más interesantes a tener en cuenta son:

- opción de visualización en pantalla o listado en impresora,
- detención de la consulta a voluntad del usuario,
- listado por registros o campos individuales o bien listado total del archivo (en este caso deberá existir la posibilidad de impresión de listados, con opciones de saltos de página correctos).

9.8.1. Operaciones sobre registros

Las operaciones de transferencia de datos a/desde un dispositivo a la memoria central se realizan mediante las instrucciones:

```
leer (<var_tipo_archivo>, lista de entrada de datos)
escribir (<var_tipo_archivo>, lista de salida de datos)
```

organización directa

```
lista de entrada de datos = numero_registro, nombre_registro
lista de salida de datos  = numero_registro, nombre_registro
```

organización secuencial

```
lista de entrada de datos = <lista_de_variables>
lista de salida de datos = <lista_de_expresiones>
```

Las operaciones de acceso a un registro y de paso de un registro a otro se realiza con las acciones **`leer`** y **`escribir`**.

9.9. PROCESAMIENTO DE ARCHIVOS SECUENCIALES (ALGORITMOS)

En un archivo secuencial los registros se insertan en el archivo en orden cronológico de llegada al soporte, es decir, un registro de datos se almacena inmediatamente a continuación del registro anterior.

Los archivos secuenciales terminan con una marca final de archivo (`FDA` o `EOF`). Cuando se tengan que añadir registros a un archivo secuencial se añadirán al final, inmediatamente por delante de las marcas fin de archivos.

Las operaciones básicas que se permiten en un archivo secuencial son: *escribir su contenido, añadir un registro al final del archivo* y *consultar sus registros*. Las demás operaciones exigen una programación específica.

Los archivos secuenciales son los que ocupan menos memoria y son útiles cuando se desconoce a priori el tamaño de los datos y se requieren registros de longitud variable. También son muy empleados para el almacenamiento de información, cuyos contenidos sufran pocas modificaciones en el transcurso de su vida útil.

Es característico de los archivos secuenciales el no poder ser utilizados simultáneamente para lectura y escritura.

9.9.1. Creación

La *creación* de un archivo secuencial es un proceso secuencial, ya que los registros se almacenan consecutivamente en el mismo orden en que se introducen en el archivo.

El método de creación de un archivo consiste en la ejecución de un programa adecuado que permita la entrada de datos al archivo desde el terminal. El sistema usual es el *interactivo*, en el que el programa solicita los datos al usuario que los introduce por teclado, al terminar se introduce una marca final de archivo, que supone el final físico del archivo.

En los archivos secuenciales, `EOF` o `FDA` es una función lógica que toma el valor *cierto* si se ha alcanzado el final de archivo y *falso* en caso contrario.

La creación del archivo requerirá los siguientes pasos:

- abrir el archivo,
- leer datos del registro,
- grabar registro,
- cerrar archivo.

El algoritmo de creación es el siguiente:

```
algoritmo  crea_sec
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ..........................
  fin_registro
  archivo_s de datos_personales: arch
var
  arch              :f
  datos_personales  :persona
inicio
  crear (f,<nombre_en_disco>)
  abrir (f,e,<nombre_en_disco>)
  leer_reg (persona)
  { utilizamos un procedimiento para no  tener que  detallar la lectura}
  mientras no ultimo_dato(persona) hacer
     escribir_f_reg (f,persona)
   //la escritura se realizará campo a campo
     leer_reg(persona)
  fin_mientras
  cerrar(f)
fin
```

Se considera que se permite la lectura y escritura en el archivo de los datos tal y como se almacenan en memoria. Un archivo de texto es un archivo secuencial en el que sólo se leen y escriben series de caracteres y no sería necesario especificar en la declaración del archivo el tipo de registros que lo constituyen, pues siempre son líneas.

9.9.2. Consulta

El proceso de búsqueda o consulta de una información en un archivo de organización secuencial se debe efectuar obligatoriamente en modo secuencial. Por ejemplo, si se desea consultar la información contenida en el registro 50, se deberán leer previamente los 49 primeros registros que le preceden en orden secuencial. En el caso de un archivo de personal, si se desea buscar un registro determinado correspondiente a un determinado empleado, será necesario recorrer —leer— todo el archivo desde el principio hasta encontrar el registro que se busca o la marca final de archivos.

Así, para el caso de un archivo de n registros, el número de lecturas de registros efectuadas son:

- mínimo 1, si el registro buscado es el primero del archivo,
- máximo n, si el registro buscado es el último o no existe dentro del archivo.

Por término medio, el número de lecturas necesarias para encontrar un determinado registro es:

$$\frac{n+1}{2}$$

El tiempo de acceso será influyente en las operaciones de lectura/escritura. Así, en el caso de una lista o vector de n elementos almacenados en memoria central puede suponer tiempos de microsegundos o nanosegundos; sin embargo, en el caso de un archivo de n registros los tiempos de acceso son de milisegundos o fracciones/múltiples de segundos, lo que supone un tiempo de acceso de 1.000 a 100.000 veces más grande una búsqueda de información en un soporte externo que en memoria central.

El algoritmo de consulta de un archivo requerirá un diseño previo de la presentación de la estructura de registros en el dispositivo de salida, de acuerdo al número y longitud de los campos.

```
algoritmo consulta_sec
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ............: ............
  fin_registro
  archivo_s de datos_personales: arch
var
  arch: f
  datos_personales:  persona
inicio
  abrir(f,l,<nombre_en_disco>)
  mientras no fda(f)hacer
       leer_f_reg(f,persona)
  fin_mientras
  cerrar(f)
fin
```

o bien:

```
inicio
  abrir(f,l,<nombre_en_disco>)
  leer_f_reg(f, persona)
  mientras no fda(f) hacer
     escribir_reg(persona)
     leer_f_reg(f,persona)
   fin_mientras
   cerrar(f)
fin
```

El uso de uno u otro algoritmo depende de cómo el lenguaje de programación detecta la marca de fin de archivo. En la mayor parte de los casos el algoritmo válido es el primero, pues la marca se detecta automáticamente con la lectura del último registro.

En el caso de búsqueda de un determinado registro, con un campo clave *x*, el algoritmo de búsqueda se puede modificar en la siguiente forma con

Consulta de un registro

Si el archivo no está ordenado:

```
algoritmo consultal_sec
tipo
  registro: datos_personales
    <tipo_dato1>:nombre_campo1
    <tipo_dato2>:nombre_campo2
    .......... : ............
  fin_registro
  archivo_s de datos_personales: arch

var
  arch            :f
  datos_personales:persona
```

```
  <tipo_dato1>    :clavebus
  lógico          :encontrado
inicio
  abrir(f,l,<nombre en_disco>)
  encontrado ← falso
  leer(clavebus)
  mientras no encontrado y no fda(f) hacer
    leer_f_reg(f, persona)
    si igual(clavebus, persona) entonces
          encontrado ← verdad
    fin_si
  fin_mientras
  si no encontrado entonces
      escribir ('No existe')
  si_no
       escribir_reg(persona)
  fin_si
  cerrar(f)
fin
```

Si el archivo está indexado en orden creciente por el campo por el cual realizamos la búsqueda se podría acelerar el proceso, de forma que no sea necesario recorrer todo el fichero para averiguar que un determinado registro no está:

```
algoritmo consulta2_sec
tipo
  registro: datos_personales
      <tipo_dato1>: nombre_campo1
      <tipo_dato2>: nombre_campo2
      ............: ............
  fin_registro
  archivo_s de datos_personales: arch
var
  arch            : f
  datos_personales: persona
  <tipo_dato1>    : clavebus
  lógico          : encontrado, pasado

inicio
  abrir(f,l,<nombre_en_disco>)
  encontrado ← falso
  pasado ← falso
  leer(clavebus)
  mientras no encontrado y no pasado y no fda(f) hacer
     leer_f_reg(f, persona)
    si igual(clavebus, persona) entonces
        encontrado ← verdad
    si_no
        si menor(clavebus, persona) entonces
            pasado ← verdad
        fin_si
    fin_si
  fin_mientras
  si no encontrado entonces
    escribir ('No existe')
```

```
    si_no
       escribir_reg(persona)
    fin_si
    cerrar(f)
fin
```

9.9.3. Actualización

La actualización de un archivo supone:

- añadir nuevos registros *(altas)*,
- modificar registros ya existentes *(modificaciones),*
- borrar registros *(bajas).*

Altas

La operación de dar de alta un determinado registro es similar a la operación de añadir datos a un archivo.

```
algoritmo añade_sec
tipo
    registro: datos_personales
        <tipo_dato1>: nombre_campo1
        <tipo_dato2>: nombre_campo2
        ...........:............
    fin_registro
    archivo_s de datos_personales:arch
var
  arch            : f
  datos_personales: persona
inicio
  abrir(f, e,<nombre_en_disco>)
  leer_reg(persona)
  mientras no ultimo_dato(persona) hacer
     escribir_f_reg (f,persona)
     leer_reg (persona)
  fin_mientras
  cerrar
fin
```

Bajas

Existen dos métodos para dar de baja un registro:

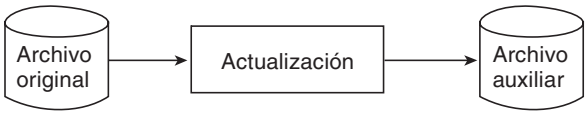
1. Se utiliza un archivo transitorio.
2. Almacenar en un array (vector) todos los registros del archivo, señalando con un indicador o bandera *(flag)* el registro que se desea dar de baja.

Método 1

Se crea un segundo archivo auxiliar, también secuencial, copia del que se trata de actualizar. Se lee el archivo completo registro a registro y en función de su lectura se decide si el registro se debe dar de baja o no.

Si el registro se va a dar de baja, se omite la escritura en el archivo auxiliar o transitorio. Si el registro no se va a dar de baja, este registro se escribe en el archivo auxiliar.

Tras terminar la lectura del archivo original, se tendrán dos archivos: *original* (o maestro) y *auxiliar*.



El proceso de bajas del archivo concluye cambiando el nombre del archivo auxiliar por el de maestro y borrando previamente el archivo maestro original.

```
algoritmo bajas_s
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ............:.............
  fin_registro
  archivo_s de datos_peersonales:arch
var
  arch            :f, faux
  datos_personales: persona, personaaux
  lógico          :encontrado
inicio
  abrir(f,l, 'antiguo')
  crear(faux, 'nuevo')
  abrir(faux, e, 'nuevo')
  leer(personaaux.nombre_campo1)
  encontrado ← falso
  mientras no fda (f) hacer
    leer_f_reg (f, persona)
    si personaaux.nombre_campo1 = persona.nombre_campo1 entonces
      encontrado ← verdad
    si_no
       escribir_f_reg (faux, persona)
    fin_si
fin_mientras
  si no encontrado entonces
    escribir ('no esta')
  fin_si
  cerrar (f, faux)
  borrar ('antiguo')
  renombrar ('nuevo', 'antiguo')
fin
```

Método 2

Este procedimiento consiste en señalar los registros que se desean dar de baja con un indicador o bandera; estos registros no se graban en el nuevo archivo secuencial que se crea sin los registros dados de baja.

Modificaciones

El proceso de modificación de un registro consiste en localizar este registro, efectuar dicha modificación y a continuación reescribir el nuevo registro en el archivo. El proceso es similar al de bajas:

```
algoritmo modificacion_sec
tipo
  registro: datos_personales
    <tipo_dato1>: nombre_campo1
    <tipo_dato2>: nombre_campo2
    ............:.............
  fin
  archivo_s de datos_personales: arch
var
  arch            : f, faux
  datos_personales: persona, personaaux
  lógico          : encontrado
inicio
  abrir(f, l, 'antiguo')
  crear(faux, 'nuevo')
  abrir(faux, e, 'nuevo')
  leer(personaaux.nombre_campo1)
  encontrado ← falso
  mientras_no fda(f) hacer
    leer_f_reg (f, persona)
    si personaaux.nombre_campo1=persona.nombre_campo1 entonces
        encontrado ← verdad
        modificar (persona)
    fin_si
    escribir_f_reg (faux, persona)
  fin_mientras
  si no encontrado entonces
               escribir ('no esta')
  fin_si
  cerrar(f, faux)
  borrar('antiguo')
  renombrar ('nuevo', 'antiguo')
fin
```

El subprograma de modificación de su registro consta de unas pocas instrucciones en las que se debe introducir por teclado el registro completo con indicación de todos sus campos o, por el contrario, el campo o campos que se desea modificar. El subprograma en cuestión podría ser:

```
procedimiento modificar(E/S datos_personales: persona)
var carácter: opcion
  entero    : n
inicio
  escribir('R.- registro completo)
  escribir('C.- campos individuales')
  escribir('elija opcion:')
  leer(opcion)
  según_sea opcion hacer
    'R'
          visualizar(persona)
          leer_reg(persona)
    'C'
          presentar(persona)
          solicitar_campo(n)
          introducir_campo(n, persona)
  fin_según
fin_procedimiento
```

9.10. PROCESAMIENTO DE ARCHIVOS DIRECTOS (ALGORITMOS)

Se dice que un archivo es aleatorio o directo cuando cualquier registro es directamente accesible mediante la especificación de un índice, que da la posición del registro con respecto al origen del fichero. Los archivos aleatorios o directos tienen una gran rapidez para el acceso comparados con los secuenciales; los registros son fáciles de referenciar —número de orden del registro—, lo que representa una gran facilidad de mantenimiento.

La lectura/escritura de un registro es rápida, ya que se accede directamente al registro y no se necesita recorrer los anteriores.

9.10.1. Operaciones con archivos directos

Las operaciones con archivos directos son las usuales, ya vistas anteriormente.

Creación

El proceso de creación de un archivo directo o aleatorio consiste en ir introduciendo los sucesivos registros en el soporte que los va a contener y en la dirección obtenida, resultante del algoritmo de conversión. Si al introducir un registro se encuentra ocupada la dirección, el nuevo registro deberá ir a la zona de sinónimos o de excedentes.

```
algoritmo crea_dir
  tipo
    registro: datos_personales
      <tipo_dato1> : nombre_campo1
      ..........  : ............
      <tipo_datoN> : nombre_campoN
       ..........  : ............
    fin_registro
    archivo_d de datos_personales: arch
  var
    arch             : f
    datos_personales : persona
  inicio
    crear(f,<nombre_en_disco>)
    abrir(f,l/e,<nombre_en_disco>)
    ........................
    { las operaciones pueden variar con arreglo al modo como
      pensemos trabajar posteriormente con el archivo
      (posicionamiento directo en un determinado registro,
      transformación de clave, indexación) }
    ........................
    cerrar(f)
  fin
```

En los registros de un archivo directo se suele incluir un campo —ocupado— que pueda servir para distinguir un registro dado de baja o modificado de un alta o de otro que nunca contuvo información.

Dentro del proceso de creación del archivo podríamos considerar una inicialización de dicho campo en cada uno de los registros del archivo directo.

```
algoritmo crea_dir
const
    max = <valor>
tipo
  registro: datos_personales
     <tipo_dato1>: cod
```

```
      <tipo_dato2>: ocupado
      .......... : ............
      <tipo_daton>: nombre_campon
      .......... : ............
  fin_registro
  archivo_d de datos_personales: arch
var
  arch             : f
  datos_personales : persona
inicio
  crear(f,<nombre_en_disco>)
  abrir(f,l/e,<nombre_en_disco>)
  desde i ← 1 hasta Max hacer
     persona.ocupado ← ' '
     escribir(f, i, persona)
  fin_desde
  cerrar(f)
fin
```

Altas

La operación de altas en un archivo directo o aleatorio consiste en ir introduciendo los sucesivos registros en una determinada posición, especificada a través del índice. Mediante el índice nos posicionaremos directamente sobre el byte del fichero que se encuentra en la posición `(indice - 1) * tamaño_de(<tipo_registros_del_archivo>)` y escribiremos allí nuestro registro.

Tratamiento por transformación de clave

El método de transformación de clave consiste en transformar un número de orden (clave) en direcciones de almacenamiento por medio de un algoritmo de conversión.

Cuando las altas se realizan por el método de transformación de clave, la dirección donde introducir un determinado registro se conseguirá por la aplicación a la clave del algoritmo de conversión (HASH). Si encontráramos que dicha dirección ya está ocupada, el nuevo registro deberá ir a la zona de sinónimos o de excedentes.

```
algoritmo altas_dir_trcl
const
  findatos = <valor1>
  max      = <valor2>
tipo
  registro: datos_personales
    <tipo_dato1>: cod
    <tipo_dato2>: ocupado
    .......... : ............
    <tipo_daton>: nombre_campon
    .......... : ............
  fin_registro
  archivo_d de datos_personales: arch
var
  arch              : f
  datos_personales  : persona, personaaux
  lógico            : encontradohueco
  entero            : posi
inicio
  abrir(f,l/e,<nombre_en_disco>)
  leer(personaaux.cod)
  posi ← HASH(personaaux.cod)
```

```
  leer(f, posi, persona)
  si persona.ocupado = '*' entonces
      encontradohueco ← falso
      posi ← findatos
      mientras posi < Max y no encontradohueco hacer
        posi ← posi + 1
        leer(f, posi, persona)
        si persona.ocupado <> '*' entonces
          encontradohueco ← verdad
        fin_si
      fin_mientras
  si_no
      encontradohueco ← verdad
  fin_si
  si encontradohueco entonces
    leer_otros_campos(personaaux)
    persona ← personaaux
    persona.ocupado ← '*'
    escribir(f, posi, persona)
  si_no
    escribir('no está')
  fin_si
  cerrar(f)
fin
```

Consulta

El proceso de consulta de un archivo directo o aleatorio es rápido y debe comenzar con la entrada del índice correspondiente al registro que deseamos consultar.

El índice permitirá el posicionamiento directo sobre el byte del fichero que se encuentra en la posición

```
(indice - 1) * tamaño_de(<var_de_tipo_registros_del_fichero>)
```

```
algoritmo consultas_dir
const
  max      = <valor1>
tipo
  registro: datos_personales
  {Cuando el código coincide con el índice o posición del
  registro en el archivo, no resulta necesario su
  almacenamiento }
   <tipo_dato1>: ocupado
   .......... : ............
   <tipo_daton>: nombre_campon
   .......... : ............
  fin_registro
  archivo_d de datos_personales: arch
var
  arch             : f
  datos_personales : persona
  lógico           : encontrado
  entero           : posi
inicio
abrir(f,l/e,<nombre_en_disco>)
leer(posi)
```

```
si (posi >=1) y (posi <= Max) entonces
  leer(f, posi, persona)
  {como al escribir los datos marcamos el campo
  ocupado con * }
  si persona.ocupado <>'*' entonces
    {para tener garantías en esta operación es
    por lo que debemos inicializar en todos los
    registros, durante el proceso de creación, el
    campo ocupado a un determinado valor,
    distinto de *}
      encontrado ← falso
  si_no
      encontrado ← verdad
  fin_si
  si encontrado entonces
    escribir_reg(persona)
  si_no
    escribir('no está')
  fin_si
si_no
    escribir('Número de registro incorrecto')
fin_si
cerrar(f)
fin
```

Consulta. *Por transformación de clave*

Puede ocurrir que la clave o código por el que deseamos acceder a un determinado registro no coincida con la posición de dicho registro en el archivo, aunque guarden entre sí una cierta relación, pues al escribir los registros en el archivo la posición se obtuvo aplicando a la clave un algoritmo de conversión.

En este caso es imprescindible el almacenamiento de la clave en uno de los campos del registro y las operaciones a realizar para llevar a cabo una consulta serían:

— Definir clave del registro buscado.
— Aplicar algoritmo de conversión clave a dirección.
— Lectura del registro ubicado en la dirección obtenida.
— Comparación de las claves de los registros leído y buscado y, si son distintas, exploración secuencial del área de excedentes.
— Si tampoco se encuentra el registro en este área es que no existe.

```
algoritmo consultas_dir_trcl
const
  findatos = <valor1>
  max      = <valor2>
tipo
  registro: datos_personales
     <tipo_dato1>: cod
     <tipo_dato2>: ocupado
     ..........  : ............
     <tipo_daton>: nombre_campon
    ..........  : ............
  fin_registro
  archivo_d de datos_personales: arch
var
  arch              : f
  datos_personales  : persona, personaaux
```

```
  lógico            : encontrado
  entero            : posi
inicio
abrir(f,l/e,<nombre_en_disco>)
leer(personaaux.cod)
posi ← HASH(personaaux.cod)
leer(f, posi, persona)
si (persona.ocupado <>'*') o (persona.cod <> personaaux.cod) entonces
  encontrado ← falso
  posi ← Findatos
  mientras (posi < Max ) y no encontrado hacer
    posi ← posi + 1
    leer(f, posi, persona)
    si (persona.ocupado = '*' )y
       (persona.cod = personaaux.cod) entonces
      encontrado ← verdad
    fin_si
  fin_mientras
si_no
  encontrado ← verdad
fin_si
si encontrado entonces
  escribir_reg(persona)
si_no
  escribir('No está')
fin_si
cerrar(f)
fin
```

Bajas

En el proceso de bajas se considera el contenido de un campo indicador, por ejemplo, `persona.ocupado`, que, cuando existe información válida en el registro está marcado con un `*`. Para dar de baja al registro, es decir, considerar su información como no válida, eliminaremos dicho `*`. Este tipo de baja es una baja lógica.

Desarrollaremos a continuación un algoritmo que realice bajas lógicas y acceda a los registros a los que se desea dar la baja por el método de transformación de clave.

```
algoritmo bajas_dir_trcl
const
  findatos = <valor1>
  max      = <valor2>
tipo
  registro: datos_personales
     <tipo_dato1>: cod
     <tipo_dato2>: ocupado
     .......... : ..........
     <tipo_daton>: nombre_campon
     .......... : ..........
  fin_registro
  archivo_d de datos_personales: arch
var
  arch             : f
  datos_personales : persona, personaaux
  lógico           : encontrado
  entero           : posi
```

```
inicio
abrir(f,l/e,<nombre_en_disco>)
leer(personaaux.cod)
posi ← HASH(personaaux.cod)
leer(f, posi, persona)
si (persona.ocupado <>'*') o
   (persona.cod <> personaaux.cod) entonces
    encontrado ← falso
    posi ← findatos
    mientras (posi < Max) y no encontrado hacer
      posi ← posi + 1
      leer(f, posi, persona)
      si (persona.ocupado='*') y
         (persona.cod = personaaux.cod) entonces
        encontrado ← verdad
      fin_si
    fin_mientras
si_no
    encontrado ← verdad
fin_si
si encontrado entonces
  persona.ocupado ← ' '
  escribir(f, posi, persona)
si_no
  escribir('No está')
fin_si
  cerrar(f)
fin
```

Modificaciones

En un archivo aleatorio se localiza el registro que se desea modificar —mediante la especificación del índice o aplicando el algoritmo de conversión clave a dirección y, en caso necesario, la búsqueda en la zona de colisiones— se modifica el contenido y se reescribe.

```
algoritmo modificaciones_dir_trcl
const
  findatos = <valor1>
  max      = <valor2>
tipo
  registro: datos_personales
    <tipo_dato1>: cod
    <tipo_dato2>: ocupado
    .......... : ...........
    <tipo_daton>: nombre_campon
    .......... : ...........
  fin_registro
  archivo_d de datos_personales: arch
var
  arch             : f
  datos_personales : persona, personaaux
  lógico           : encontrado
  entero           : posi
inicio
  abrir(f,l/e,<nombre_en_disco>)
  leer(personaaux.cod)
```

```
    posi ← HASH(personaaux.cod)
    leer(f, posi, persona)
    if (persona.ocupado <>'*') o
        (persona.cod <> personaaux.cod) entonces
        encontrado ← falso
        posi ← findatos
        mientras posi < max y no encontrado hacer
            posi ← posi + 1
            leer(f, posi, persona)
            si (persona.ocupado = '*') y (persona.cod = personaaux.cod)
               entonces
              encontrado ← verdad
            fin_si
        fin_mientras
    si_no
        encontrado ← verdad
    fin_si
    si encontrado entonces
        leer_otros_campos(personaaux)
        personaaux.ocupado ← '*'
        escribir(f, posi, personaaux)
    si_no
        escribir('no está')
    fin_si
    cerrar(f)
fin
```

9.10.2. Clave-dirección

Con respecto a las transformaciones clave-dirección deberemos realizar aún algunas consideraciones.

En un soporte direccionable —normalmente un disco—, cada posición se localiza por su dirección absoluta —número de pista y número de sector en el disco—. Los archivos directos manipulan direcciones relativas en lugar de absolutas, lo que hará al programa independiente de la posición absoluta del archivo en el soporte. Los algoritmos de conversión de clave transformarán las claves en direcciones relativas. Suponiendo que existen `N` posiciones disponibles para el archivo, los algoritmos de conversión de clave producirán una dirección relativa en el rango 1 a `N` por cada valor de la clave.

Existen varias técnicas para obtener direcciones relativas. En el caso en que dos registros distintos produzcan la misma dirección, se dice que se produce una colisión o sinónimo.

9.10.3. Tratamiento de las colisiones

Las colisiones son inevitables y, como se ha comentado, se originan cuando dos registros de claves diferentes producen la misma dirección relativa. En estos casos las colisiones se pueden tratar de dos formas diferentes.

Supongamos que un registro e1 produce una dirección d1 que ya está ocupada. ¿Dónde colocar el nuevo registro? Existen dos métodos básicos:

- Considerar una zona de excedentes y asignar el registro a la primera posición libre en dicha zona. Fue el método aplicado en los algoritmos anteriores.
- Buscar una nueva dirección libre en la zona de datos del archivo.

9.10.4. Acceso a los archivos directos mediante indexación

La indexación es una técnica para el acceso a los registros de un archivo. En esta técnica el archivo principal de registros está suplementado por uno o más índices. Los índices pueden ser archivos independientes o un array que se

carga al comenzar en la memoria del ordenador, en ambos casos estarán formados por registros con los campos código o clave y posición o número de registro.

El almacenamiento de los índices en memoria permite encontrar los registros más rápidamente que cuando se trabaja en disco.

Cuando se utiliza un archivo indexado se localizan los registros en el índice a través del campo clave y éste retorna la posición del registro en el archivo principal, directo.

Las operaciones básicas a realizar con un archivo indexado son:

— Crear las zonas de índice y datos como archivos vacíos originales.
— Cargar el archivo índice en memoria antes de utilizarlo.
— Reescribir el archivo índice desde memoria después de utilizarlo.
— Añadir registros al archivo de datos y al índice.
— Borrar registros.
— Actualizar registros en el archivo de datos.

Consulta

Como ejemplo veamos la operación de consulta de un registro

```
algoritmo consulta_dir_ind
const
  max = <valor>
tipo
  registro: datos_personales
    <tipo_dato1>: cod
    <tipo_dato2>: nombre_campo2
    .......... : ...........
    <tipo_daton>: nombre_campon
    .......... : ...........
  fin_registro
  registro: datos_indice
    <tipo_dato1>: cod
    entero      : posi
  fin_registro
  archivo_d de datos_personales: arch
  archivo_d de datos_indice    : ind
  array[1..max] de datos_indice: arr
var
  arch              : f
   ind              : t
   arr              : a
  datos_personales  : persona
   entero           : i, n, central
   <tipo_dato1>     : cod
   lógico           : encontrado
inicio
  abrir(f,l/e,<nombre_en_disco1>)
   abrir(t,l/e,<nombre_en_disco2>)
   n ← LDA(t)/tamaño_de(datos_indice)
  desde i ← 1 hasta n hacer
      leer(t,i,a[i])
  fin_desde
  cerrar(t)
  {Debido a la forma de efectuar las altas el archivo
   índice siempre tiene sus registros ordenados por el campo cod }
  leer(cod)
  busqueda_binaria(a, n, cod, central, encontrado)
```

```
  { el procedimiento de búsqueda_binaria en un array será
   desarrollado en capítulos posteriores del libro}
  si encontrado entonces
      leer(f, a[central].posi, persona)
      escribir_reg(persona)
  si_no
      escribir('no está')
  fin_si
  cerrar(f)
fin
```

Altas

El procedimiento empleado para dar las altas en el archivo anterior podría ser el siguiente:

```
procedimiento altas(E/S arr: a   E/S entero: n)
  var
  reg           : persona
  entero        : p
  lógico        : encontrado
  entero        : num
inicio
  si n = max entonces
      escribir('lleno')
 si_no
      leer_reg(persona)
      encontrado ← falso
      busqueda_binaria(a, n, persona.cod, p, encontrado)
      si encontrado entonces
        escribir('Clave duplicada')
      si_no
        num ← LDA(f)/tamaño_de(datos_personales) + 1
        {Insertamos un nuevo registro en la tabla
          sin que pierda su ordenación }
        alta_indice(a, n, p, persona.cod, num)
        n ← n + 1
        {Escribimos el nuevo registro al final del
          archivo principal }
        escribir(f, num, persona)
      fin_si
  fin_si
  { en el programa principal, al terminar, crearemos de
    nuevo el archivo índice a partir de los registros
    almacenados en el array a }
fin_procedimiento
```

9.11. PROCESAMIENTO DE ARCHIVOS SECUENCIALES INDEXADOS

Los archivos de organización secuencial indexada contienen tres áreas: un área de datos que agrupa a los registros, un área índice que contiene los niveles de índice y una zona de desbordamiento o excedentes para el caso de actualizaciones con adición de nuevos registros.

Los registros han de ser grabados obligatoriamente en orden secuencial ascendente por el contenido del campo clave y, simultáneamente a la grabación de los registros, el sistema crea los índices.

Una consideración adicional con respecto a este tipo de organización es que es posible usar más de una clave, hablaríamos así de la clave primaria y de una o más secundarias. El valor de la clave primaria es la base para la po-

sición física de los registros en el archivo y debe ser única. Las claves secundarias pueden ser o no únicas y no afectan al orden físico de los registros.

9.12. TIPOS DE ARCHIVOS: CONSIDERACIONES PRÁCTICAS EN C/C++ Y JAVA

Los archivos se pueden clasificar en función de determinadas características. Entre ellas las más usuales son: por el tipo de acceso o por la estructura de la información del archivo.

Dirección del flujo de datos

Los archivos se clasifican en función del flujo de los datos o por el modo de acceso a los datos del archivo. En función de la dirección del flujo de los datos son de:

- *Entrada.* Aquellos cuyos datos se leen por parte del programa (archivos de lectura).
- *Salida.* Archivos que escribe el programa (archivos de escritura).
- *Entrada/Salida.* Archivos en los que se puede leer y escribir.

La determinación del tipo de archivo se realiza en el momento de la creación del archivo.

Tipos de acceso

Los archivos se clasifican en:

- *Secuenciales*. El orden de acceso a los datos es secuencial; primero se accede al primer elemento, luego al segundo y así sucesivamente.
- *Directos (aleatorios).* El acceso a un elemento concreto del archivo es directo. Son similares a las tablas.

Estructura de la información

Los archivos guardan información en formato binario y se distribuye en una secuencia o flujo de bytes. Teniendo en cuenta la información almacenada los archivos se clasifican en:

- ***Texto.*** En estos archivos se guardan solamente ciertos caracteres imprimibles, tales como letras, números y signos de puntuación, salto de línea, etc. Están permitidos ciertos rangos de valores para cada *byte*. En un archivo de texto no está permitido el *byte* de final de archivo y si existe no se puede ver más allá de la posición donde está el *byte.*
- ***Binarios.*** Contienen cualquier valor que se pueda almacenar en un *byte.*

El tipo de información almacenada en los archivos se define a la hora de abrirlos (para lectura, escritura). Con posterioridad, cada operación leerá los bytes correspondientes al tipo de datos.

Un archivo de texto es un caso particular de archivo de organización secuencial y es una serie continua de caracteres que se pueden leer uno tras otro. Cada registro de un archivo de texto es del tipo de cadena de caracteres.

El tratamiento de archivos de texto es elemental y en el caso de lenguajes como Pascal es posible detectar lecturas de caracteres especiales como final de archivo o final de línea.

9.12.1. Archivos de texto

Los archivos de texto también se denominan archivos ASCII y son legibles por los usuarios o programadores. Los terminales, los teclados y las impresoras tratan con datos carácter. Así, cuando se desea escribir un número como "1234" en la pantalla se debe convertir a cuatro caracteres ("1", "2", "3", "4") y ser escritos en el terminal.

De modo similar cuando se lee un número de teclado, los datos se deben convertir de caracteres a enteros. En el caso del lenguaje C++ esta operación se realiza con el operador >>. Las computadoras trabajan con datos binarios. Cuando se leen números de un archivo ASCII, el programa debe procesar los datos carácter a través de una rutina de conversión, lo que entraña grandes recursos. Los archivos binarios, por el contrario, no requieren conversión e incluso ocupan menos espacio que los archivos ASCII; su gran inconveniente es que los archivos binarios no se pueden imprimir directamente en una impresora ni visualizar en un terminal.

Los archivos ASCII son *portables* (en la mayoría de los casos) y se pueden mover de una computadora a otra sin grandes problemas. Sin embargo, los archivos binarios son prácticamente no portables; a menos que sea un programador experto es casi imposible hacer portable un archivo binario.

9.12.2. Archivos binarios

Los archivos binarios contienen cualquier valor que se puede almacenar en un *byte.* En estos archivos el final del archivo no se almacena como un *byte* concreto. Los archivos binarios se escriben copiando una imagen del contenido de un segmento de la memoria al disco y por consiguiente los valores numéricos aparecen como unos caracteres extraños que se corresponden con la codificación de dichos valores en la memoria de la computadora, aunque aparentemente son prácticamente indescifrables para el programador o el usuario.

Cuando se intenta abrir un archivo binario con el editor aparecerán secuencias de caracteres tales como:

```
E#@%Âa^^...
```

¿Cuál es el archivo más recomendable para utilizar? En la mayoría de los casos, el ASCII es el mejor. Si se tienen pequeñas a medianas cantidades de datos el tiempo de conversión no afecta seriamente a su programa. Por otra parte los archivos ASCII también facilitan la verificación de los datos. Por el contrario, sólo cuando se utilizan grandes cantidades de datos los problemas de espacio y rendimiento, normalmente, aconsejarán utilizar formatos binarios.

Los archivos de texto se suelen denominar con la extensión `.txt`, mientras que los archivos binarios suelen tener la extensión `.dat`. Los archivos de texto son muy eficientes para intercambiar datos entre aplicaciones y para proporcionar datos de entrada de programas que se deban ejecutar varias veces; por el contrario, son poco eficientes para manejar grandes volúmenes de información o bases de datos. Por otra parte, todos los archivos binarios permiten acceso directo, lo cual es muy útil para manejar grandes archivos o bases de datos, ya que se puede ir directamente a leer el registro n sin tener que leer antes el primero, el segundo,…, el n – 1 registros anteriores

9.12.3. Lectura y escritura de archivos

Las operaciones típicas sobre un archivo son: creación, lectura y escritura. En el caso de C++ los archivos se manipulan mediante un tipo de objeto flujo. Normalmente los objetos que se usan para tratar con archivos se llaman **archivos lógicos** y **archivos físicos** son aquellos que almacenan realmente la información en disco (o dispositivos de memoria secundaria correspondiente: disco duro, discos ópticos, memorias *flash,* etc.).

En el caso del lenguaje C++, para todas las operaciones con archivo se necesita utilizar la biblioteca de cabecera fstream.h por lo que es preciso que los programas inserten la sentencia

```
#include <fstream.h>
```

o bien en el caso de ANSI C++ estándar

```
#include <fstream >
using namespace std;
```

En general, todo tratamiento de un archivo consta de tres pasos importantes:

- *Apertura del archivo.* El modo de implementar la operación dependerá de si un archivo es de lectura o escritura.
- *Acceso al archivo.* En esta etapa se llega o imprimen los datos.
- *Cierre del archivo.* Actualiza el archivo y se elimina la información no significativa.

ACTIVIDADES DE PROGRAMACIÓN RESUELTAS

9.1. *Escribir un algoritmo que permita la creación e introducción de los primeros datos en un archivo secuencial,* PERSONAL*, que deseamos almacene la información mediante registros de siguiente tipo.*

```
tipo
  registro:   datos_personales
     <tipo_dato1>   :   nombre_campo1
     <tipo_dato2>   :   nombre_campo2
     ............   :   ............
  fin_registro
```

Análisis del problema

Tras la creación y apertura en modo conveniente del archivo, el algoritmo solicitará la introducción de datos por teclado y los almacenará de forma consecutiva en el archivo.

Se utilizará una función, `ultimo_dato(persona)`, para determinar el fin en la introducción de datos.

Diseño del algoritmo

```
algoritmo  Ejercicio_9_1
tipo
  registro:   datos_personales
     <tipo_dato1>   :   nombre_campo1
     <tipo_dato2>   :   nombre_campo2
     ............   :   ............
  fin_registro
  archivo_s de  datos_personales: arch
var
  arch              :   f
  datos_personales  :   persona
inicio
  crear (f, 'Personal')
  abrir (f,e,'Personal')
  llamar_a leer_reg (persona)
  //  Procedimiento para la lectura de un
  //  registro campo a campo
  mientras no ultimo_dato(persona) hacer
    llamar_a escribir_f_reg (f,  persona)
    //  Procedimiento auxiliar, no desarrollado, para la
    //  escritura en el archivo del registro campo a campo
    llamar_a leer_reg(persona)
  fin_mientras
  cerrar (f)
fin
```

9.2. *Supuesto que deseamos añadir nueva información al archivo* PERSONAL*, anteriormente creado, diseñar el algoritmo correspondiente.*

Análisis del problema

Al abrir el archivo, para escritura se coloca el puntero de datos al final del mismo, permitiéndonos, con un algoritmo similar al anterior, la adición de nueva información al final del mismo.

Diseño del algoritmo

```
algoritmo  Ejercicio_9_2
tipo
  registro:  datos_personales
     <tipo_dato1> : nombre_campo1
```

```
      <tipo_dato2> : nombre_campo2
      ............ : ............
  fin_registro
  archivo_s de  datos_personales: arch
var
  arch             : f
  datos_personales : persona
inicio
  abrir (f,e,'PERSONAL')
  llamar_a leer_reg (persona)
  mientras no ultimo_dato (persona) hacer
     llamar_a escribir_f_reg (f, persona)
     llamar_a leer_reg (persona)
  fin_mientras
  cerrar (f)
fin
```

9.3. *Diseñar un algoritmo que muestre por pantalla el contenido de todos los registros del archivo* `PERSONAL`.

Análisis del problema

Se debe abrir el archivo para lectura y, repetitivamente, leer los registros y mostrarlos por pantalla hasta detectar el fin de fichero.

Se considera que la función `FDA(id_arch)` detecta el final de archivo con la lectura de su último registro.

Diseño del algoritmo

```
algoritmo  Ejercicio_9_3
tipo
  registro:  datos_personales
     <tipo_dato1>: nombre_campo1
     <tipo_dato2>: nombre_campo2
     ............: ............
  fin_registro
  archivo_s de datos_personales: arch
var
  arch             : f
  datos_personales : persona
inicio
  abrir (f,l,'PERSONAL')
  mientras no fda (f) hacer
     llamar_a leer_f_reg (f, persona)
     llamar_a escribir_reg (persona)
  fin_mientras
  cerrar (f)
fin
```

Si se considera la existencia de un registro especial que marca el fin de archivo, la función `FDA(id_arch)` se activaría al leer este registro y es necesario modificar algo nuestro algoritmo.

```
inicio
  abrir (f,l,'PERSONAL')
  llamar_a leer_f_reg (f, persona)
  mientras no fda (f) hacer
     llamar_a escribir_reg (persona)
     llamar_a leer_f_reg (f, persona)
  fin_mientras
  cerrar (f)
fin
```

9.4. *Una librería almacena en un archivo secuencial la siguiente información sobre cada uno de sus libros:* `CODIGO`, `TITULO`, `AUTOR` *y* `PRECIO`.

El archivo está ordenado ascendentemente por los códigos de los libros —de tipo cadena—, que no pueden repetirse.

Se precisa un algoritmo con las opciones:

1. *Insertar:* *Permitirá insertar nuevos registros en el archivo, que debe mantenerse ordenado en todo momento.*
2. *Consulta:* *Buscará registros por el campo* `CODIGO`.

Análisis del problema

El algoritmo comenzará presentando un menú de opciones a través del cual se haga posible la selección de un procedimiento u otro.

`Insertar`: Para poder colocar el nuevo registro en el lugar adecuado, sin que se pierda la ordenación inicial, se necesita utilizar un archivo auxiliar. En dicho auxiliar se van copiando los registros hasta llegar al punto donde debe colocarse el nuevo, entonces se escribe y continua con la copia de los restantes registros.

`Consulta`: Como el archivo está ordenado y los códigos no repetidos, el proceso de consulta se puede acelerar. Se recorre el archivo de forma secuencial hasta encontrar el código buscado, o hasta que éste sea menor que el código del registro que se acaba de leer desde el archivo, o bien, si nada de esto ocurre, hasta el fin del archivo.

Cuando el código buscado es menor que el código del registro que se acaba de leer desde el archivo, se puede deducir que de ahí en adelante ese registro ya no podrá estar en el fichero, por tanto, se puede abandonar la búsqueda.

Diseño del algoritmo

```
algoritmo  Ejercicio_9_4
tipo
  registro : reg
    cadena : cod
    cadena : titulo
    cadena : autor
    entero : precio
  fin_registro
  archivo_s de reg : arch
var
  entero : op

inicio
  repetir
    escribir( 'MENU')
    escribir( '1.- INSERTAR')
    escribir( '2.- CONSULTA')
    escribir( '3.- FIN')
    escribir( 'Elija opcion ')
    leer (op )
    según_sea op hacer
       1 : llamar_a insertar
       2 : llamar_a consulta
    fin_según
  hasta_que op = 3
fin

procedimiento insertar
var
  arch     : f, f2
  reg      : rf,r
```

```
  lógico   : escrito
  carácter : resp
inicio
  repetir
    abrir (f,1,'Libros.dat')
    crear (f2, 'Nlibros.dat')
    abrir (f2,e, 'Nlibros.dat')
    escribir ('Deme el codigo')
    leer (r.cod)
    escrito ← falso
    mientras no FDA(f) hacer
      llamar_a leer_arch_reg ( f, rf)
      si rf.cod > r.cod y no escrito entonces
      // si se lee del archivo un registro con codigo
      // mayor que el nuevo y este aun no se
      // ha escrito, es el momento de insertarlo
         escribir( 'Deme otros campos ')
         llamar_a completar ( r )
         llamar_a escribir_arch_reg ( f2, r )
         escrito ← verdad
         // Se debe marcar que se ha escrito
         // para que no siga insertandose, desde aqui
         // en adelante
      si_no
         si rf.cod = r.cod entonces
            escrito ← verdad
         fin_si
      fin_si
      llamar_a escribir_arch_reg ( f2, rf )
      // De todas formas se escribe el que
      // se lee del archivo
    fin_mientras
    si no escrito entonces
    // Si el codigo del nuevo es mayor que todos los del
    // archivo inicial, se llega al final sin haberlo
    // escrito
       escribir ('Deme otros campos')
       llamar_a completar (r)
       llamar_a escribir_arch_reg ( f2, r )
    fin_si
     cerrar (f, f2)
     borrar ( 'Libros.dat')
     renombrar ('Libros.dat', 'Libros.dat')
     escribir ('¿Seguir? (s/n) ')
     leer ( resp )
  hasta_que resp = 'n'
fin_procedimiento

procedimiento consulta
var
  reg: rf, r
  arch: f
  carácter: resp
  lógico: encontrado, pasado
inicio
  resp ← 's'
  mientras resp <> 'n' hacer
    abrir (f, 1, 'Libros.dat')
    escribir ('Deme el codigo a buscar ')
```

```
    leer ( r.cod)
    encontrado ← falso
    pasado ← falso
    mientras no FDA (f) y no encontrado y no pasado hacer
      llamar_a leer_arch_reg (f, rf)
      si r.cod = rf.cod entonces
         encontrado ← verdad
         llamar_a escribir_reg ( rf )
      si_no
         si r.cod < rf.cod entonces
            pasado ← verdad
         fin_si
      fin_si
    fin_mientras
    si no encontrado entonces
      escribir ( 'Ese libro no esta')
    fin_si
    cerrar (f)
    escribir ('¿Seguir? (s/n)')
    leer ( resp )
  fin_mientras
fin_procedimiento
```

9.5. *Diseñar un algoritmo que efectúe la creación de un archivo directo —*PERSONAL*—, cuyos registros serán del siguiente tipo:*

```
tipo
registro:  datos_personales
   <tipo_dato1> : cod        // Campo clave
   ...........  : ............
   <tipo_datoN> : nombre_campoN
fin_registro
```

y en el que, posteriormente, vamos a introducir la información empleando el método de transformación de clave.

Análisis del problema

El método de transformación de claves consiste en introducir los registros, en el soporte que los va a contener, en la dirección que proporciona el algoritmo de conversión. Su utilización obliga al almacenamiento del código en el propio registro y hace conveniente la inclusión en el registro de un campo auxiliar —ocupado— en el que se marque si el registro está o no ocupado. Durante el proceso de creación se debe realizar un recorrido de todo el archivo inicializando el campo ocupado a vacío, por ejemplo, a espacio.

Diseño del algoritmo

```
algoritmo Ejercicio_9_5
const
  Max = <valor>
tipo
  registro: datos_personales
     <tipo_dato1> : cod // Podria no ser necesario
                        // su almacenamiento, en el caso
                        // de que coincidiera con el
                        // indice
     ..... : .............
     <tipo_daton> : nombre_campon
  fin_registro
  archivo_d de datos_personales: arch
```

```
var
  arch             : f
  datos personales : persona
  entero           : i
inicio
  crear (f, 'PERSONAL')
  abrir (f,1/e, 'PERSONAL')
  desde i ← 1 hasta Max hacer
     persona.ocupado
     escribir (f, persona, i)
  fin_desde
  cerrar (f)
fin
```

9.6. *Se desea introducir información, por el método de transformación de clave, en el archivo* `PERSONAL` *creado en el ejercicio anterior, diseñar el algoritmo correspondiente.*

Análisis del problema

Como anteriormente se ha explicado, el método de transformación de claves consiste en introducir los registros, en el soporte que los va a contener, en la dirección que proporciona el algoritmo de conversión.

A veces, registros distintos, sometidos al algoritmo de conversión, proporcionan una misma dirección, por lo que se debe tener previsto un espacio en el disco para el almacenamiento de los registros que han consolidado. Aunque se puede hacer de diferentes maneras, en este caso se reserva espacio para las colisiones en el propio fichero a continuación de la zona de datos.

Se supone que la dirección más alta capaz de proporcionar el algoritmo de conversión es `Findatos` y se colocan las colisiones que se produzcan a partir de allí en posiciones consecutivas del archivo.

La inicialización a espacio del campo ocupado se realiza hasta `Max`, dando por supuesto que `Max` es mayor que `Findatos`.

Diseño del algoritmo

```
algoritmo Ejercicio_9_6
const
  Findatos = <valor1>
  Max      = <valor2>
tipo
  registro: datos_peronales
    <tipo_dato1> : cod // Podría no ser necesario
                       // su almacenamiento, en el caso
                       // de que coincidiera con el
                       // indice
    ............ : ............
    <tipo_daton> : nombre_campon
  fin_registro
  archivo_d de datos_personales: arch
var
  arch             : f
  datos personales : persona, personaaux
  lógico           : encontradohueco
  entero           : i
inicio
  abrir (f,1/e, 'PERSONAL')
  leer (personaaux.cod)
  posi ← HASH (personaaux.cod)
  // HASH es el nombre de la funcion de transformacion de
  // claves. La cual devolvera valores
  // entre 1 y Findatos, ambos inclusive
```

```
  leer(f, persona, posi)
  si persona.ocupado ='*' entonces //El '*' indica que esta
                                   //ocupado
    encontradohueco ← falso
    posi ← Findatos
    mientras posi < Max y no encontradohueco hacer
      posi ← posi + 1
      leer(f, persona, posi)
      si persona.ocupado <> '*' entonces
        encontradohueco ← verdad
      fin_si
    fin_mientras
  si_no
    encontradohueco ← verdad
  fin_si
  si encontradohueco entonces
    llamar_a leer_otros_campos (personaaux)
    persona ← personaaux
    persona.ocupado ← '*' //Al dar un alta marcaremos
                          //el campo ocupado
    escribir(f, persona, posi)
  si_no
    escribir ('No esta')
  fin_si
  cerrar (f)
fin
```

CONCEPTOS CLAVE

- Archivos de texto.
- Concepto de flujo.
- Organización de archivos.
- Organización directa
- Organización secuencial.
- Organización secuencial indexada.
- Registro físico.
- Registro lógico.

RESUMEN

Un archivo de datos es un conjunto de datos relacionados entre sí y almacenados en un dispositivo de almacenamiento externo. Estos datos se encuentran estructurados en una colección de entidades denominadas artículos o registros, de igual tipo, y que constan a su vez de diferentes entidades de nivel más bajo denominadas campos. Un archivo de texto es el que está formado por líneas, constituidas a su vez por una serie de caracteres, que podrían representar los registros en este tipo de archivos. Por otra parte, los archivos pueden ser binarios y almacenar no sólo caracteres sino cualquier tipo de información tal y como se encuentra en memoria.

1. Java y C# realizan las operaciones en archivos a través de flujos, manipulados por clases, que conectan con el medio de almacenamiento. De forma que para crear, leer o escribir un archivo se requiere utilizar una clase que defina la funcionalidad del flujo. Los flujos determinan el sentido de la comunicación (lectura, escritura o lectura/escritura), el posicionamiento directo o no en un determinado registro y la forma de leer y/o escribir en el archivo. Pueden utilizarse flujos de bytes cadenas o tipos primitivos. La personalización de flujos se consigue por asociación o encadenamiento de otros flujos con los flujos base de apertura de archivos.
2. Registro lógico es una colección de información relativa a una entidad particular. El concepto de registro es similar al de estructura desde el punto de vista de que permiten almacenar datos de tipo heterogéneo.

3. Registro físico es la cantidad más pequeña de datos que pueden transferirse en una operación de entrada/salida entre la memoria central y los dispositivos.
4. La organización de archivos define la forma en la que los archivos se disponen sobre el soporte de almacenamiento y puede ser secuencial, directa o secuencial-indexada.
5. La organización secuencial implica que los registros se almacenan unos al lado de otros en el orden en el que van siendo introducidos y que para efectuar el acceso a un determinado registro es necesario pasar por los que le preceden.
6. Los archivos de texto se consideran una clase especial de archivos secuenciales.
7. En la organización directa el orden físico de los registros puede no corresponderse con aquel en el que han sido introducidos y el acceso a un determinado registro no obliga a pasar por los que le preceden. Para poder acceder a un determinado registro de esta forma se necesita un soporte direccionable y la longitud de los registros debe ser fija.
8. La organización secuencial-indexada requiere la existencia de un área de datos, un área de índices, un área de desbordamiento o colisiones y soporte direccionable.

EJERCICIOS

9.1. Diseñar un algoritmo que permita crear un archivo AGENDA de direcciones cuyos registros constan de los siguientes campos:

```
NOMBRE
DIRECCION
CIUDAD
CODIGO POSTAL
TELEFONO
EDAD
```

9.2. Realizar un algoritmo que lea el archivo AGENDA e imprima los registros de toda la agenda.

9.3. Diseñar un algoritmo que copie el archivo secuencial AGENDA de los ejercicios anteriores en un archivo directo DIRECTO_AGENDA, de modo que cada registro mantenga su posición relativa.

9.4. Se dispone de un archivo indexado denominado DIRECTORIO, que contiene los datos de un conjunto de personas y cuya clave es el número del DNI. Escribir un algoritmo capaz de realizar una consulta de un registro. Si no se encuentra el registro se emite el correspondiente mensaje de ERROR.

9.5. Se dispone de un archivo STOCK correspondiente a la existencia de artículos de un almacén y se desea señalar aquellos artículos cuyo nivel está por debajo del mínimo y que visualicen un mensaje "hacer pedido". Cada artículo contiene un registro con los siguientes campos: número del código del artículo, nivel mínimo, nivel actual, proveedor, precio.

9.6. El director de un colegio desea realizar un programa que procese un archivo de registros correspondiente a los diferentes alumnos del centro a fin de obtener los siguientes datos:

- Nota más alta y número de identificación del alumno correspondiente.
- Nota media por curso.
- Nota media del colegio.

NOTA: *Si existen varios alumnos con la misma nota más alta, se deberán visualizar todos ellos.*

9.7. Diseñar un algoritmo que genere un archivo secuencial BIBLIOTECA, cuyos registros contienen los siguientes campos:

```
TITULO
AUTOR
EDITORIAL
AÑO DE EDICION
ISBN
NUMERO DE PAGINAS
```

9.8. Diseñar un algoritmo que permita modificar el contenido de alguno de los registros del archivo secuencial BIBLIOTECA mediante datos introducidos por teclado.

CAPÍTULO 10

Ordenación, búsqueda e intercalación

10.1. Introducción
10.2. Ordenación
10.3. Búsqueda
10.4. Intercalación

ACTIVIDADES DE PROGRAMACIÓN RESUELTAS
CONCEPTOS CLAVE
RESUMEN
EJERCICIOS

INTRODUCCIÓN

Las computadoras emplean una gran parte de su tiempo en operaciones de *búsqueda*, *clasificación* y *mezcla de datos*. Las operaciones de cálculo numérico y sobre todo de gestión requieren normalmente operaciones de clasificación de los datos: ordenar fichas de clientes por orden alfabético, por direcciones o por código postal. Existen dos métodos de ordenación: *ordenación interna* (de **arrays**, **arreglos**) y *ordenación externa* (archivos). Los arrays se almacenan en la memoria interna o central, de acceso aleatorio y directo, y por ello su gestión es rápida. Los *archivos* se sitúan adecuadamente en dispositivos de almacenamiento externo que son más lentos y basados en dispositivos mecánicos: cintas y discos magnéticos. Las técnicas de ordenación, búsqueda y mezcla son muy importantes y el lector deberá dedicar especial atención al conocimiento y aprendizaje de los diferentes métodos que en este capítulo se analizan.