Fundamentos de Programación II

2. Archivos

Luis Rodríguez Baena (luis.rodriguez@upsam.es)

Universidad Pontificia de Salamanca
Escuela Superior de Ingeniería y Arquitectura

Concepto de archivo

- Necesidad de almacenar los datos en dispositivos de memoria auxiliares (discos) que mantengan la información de forma permanente.
- Necesidad de poder tratar grandes cantidades de información de forma fraccionada de forma que la memoria central pueda tratarlas.
- Limitaciones de los arrays.
  - Carácter finito.
    - ✓ Definen su tamaño en tiempo de compilación y no se puede modificar a lo largo de la ejecución de un programa.
    - ✓ El tamaño de la memoria es limitado.
  - Los archivos tienen una *cardinalidad virtualmente infinita*.
    - ✓ Toman o liberan espacio de almacenamiento en tiempo de ejecución.
    - ✓ En realidad el tamaño está limitado por el tamaño de los dispositivos de almacenamiento y el sistema operativo.
  - Carácter temporal.
    - ✓ Los arrays guardan información no permanente.
      - Su tiempo de vida máximo es el tiempo de vida de la aplicación en memoria.
  - Los archivos permiten almacenar información de forma permanente.
    - ✓ No es necesario carga la información.
    - ✓ Es posible intercambiar información de forma fácil entre programas.

Concepto de archivo (II)

- El tipo de dato archivo.
  - Archivo: colección ordenada de datos homogéneos *almacenada en un dispositivo externo.*
    - ✓ El dispositivo podrá ser un soporte de almacenamiento auxiliar (disco) o cualquier flujo de información de entrada o salida (teclado, impresora, pantalla, puertos serie, memoria, etc.).
  - Es un dato estático.
    - ✓ Aunque el tamaño de un archivo en disco puede variar en función de la información contenida, desde el punto de vista de la memoria, se considera un dato estático.
    - ✓ Un archivo ocupará en memoria un tamaño fijo, correspondiente al *buffer* empleado para la comunicación entre el disco y la memoria principal.
      - ○ A diferencia de los arrays, sólo se almacenará en la memoria principal un fragmento del archivo almacenado en disco.

Estructura lógica

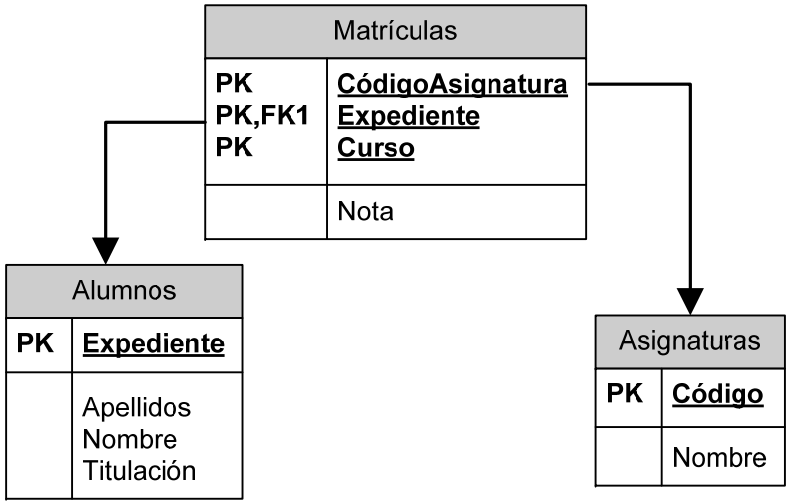
- Hace referencia a la forma en que un algoritmo ve al archivo.
- Un archivo está compuesto por una colección de datos homogéneos.
  - Cada vez que se realiza una operación de entrada o salida en un archivo, se lee o se escribe uno de esos datos.
    - ✓ A la cantidad mínima de información que se puede leer o escribir en un archivo se le denomina **registro**.
  - El concepto de registro es distinto de la estructura de datos de tipo registro.
    - ✓ La información que se lee o escribe podrán ser enteros, reales, cadenas, caracteres, o estructuras de datos de tipo registro.
      - Habrá archivos de enteros, de caracteres, de cadenas (si el registro son datos simples) o de personas, de estudiantes o de cualquier dato de tipo registro o estructura.

Estructura lógica (II)

- Cuando el tipo base de un archivo es un dato de tipo registro (una estructura), se podrá descomponer en datos más elementales: **campos**.
  - Por ejemplo, un archivo de personas podrá estar compuesto por registros de tipo persona con información sobre su DNI, sus apellidos, su nombre, su provincia de residencia o su teléfono.
  - Dentro de los campos pueden existir determinados **campos clave**.
    - ✓ Campos que identifican un registro dentro de un archivo.
      - Clave primaria (*primary key, PK*).
        - Identifica un registro de forma unívoca.
        - No admite duplicados.
        - Por ejemplo, el DNI.
      - Claves alternativas o secundarias (*foreign key, FK*).
        - Permiten realizar otros procesos como ordenaciones por distintos campos o búsquedas alternativas.
        - Pueden tener o no duplicados.
        - Por ejemplo, la provincia de residencia o los apellidos.
      - Claves compuestas.
        - Tanto las claves primarias como las alternativas se pueden obtener por la combinación de distintos campos
        - Por ejemplo, una clave posible podría estar formada por los apellidos y el nombre de la persona.

Estructura lógica (III)

- Base de datos.
  - De forma simplificada podemos considerar una base de datos como una colección de archivos.
  - Conjuntos de información relacionada.
  - Cada conjunto (tabla) estará a su vez compuesta de registros (tuplas o filas) y campos (columnas).
  - Los conjuntos estarán relacionados por claves externas que permiten acceder a la información de una tabla con los datos de otra: bases de datos relacionales.



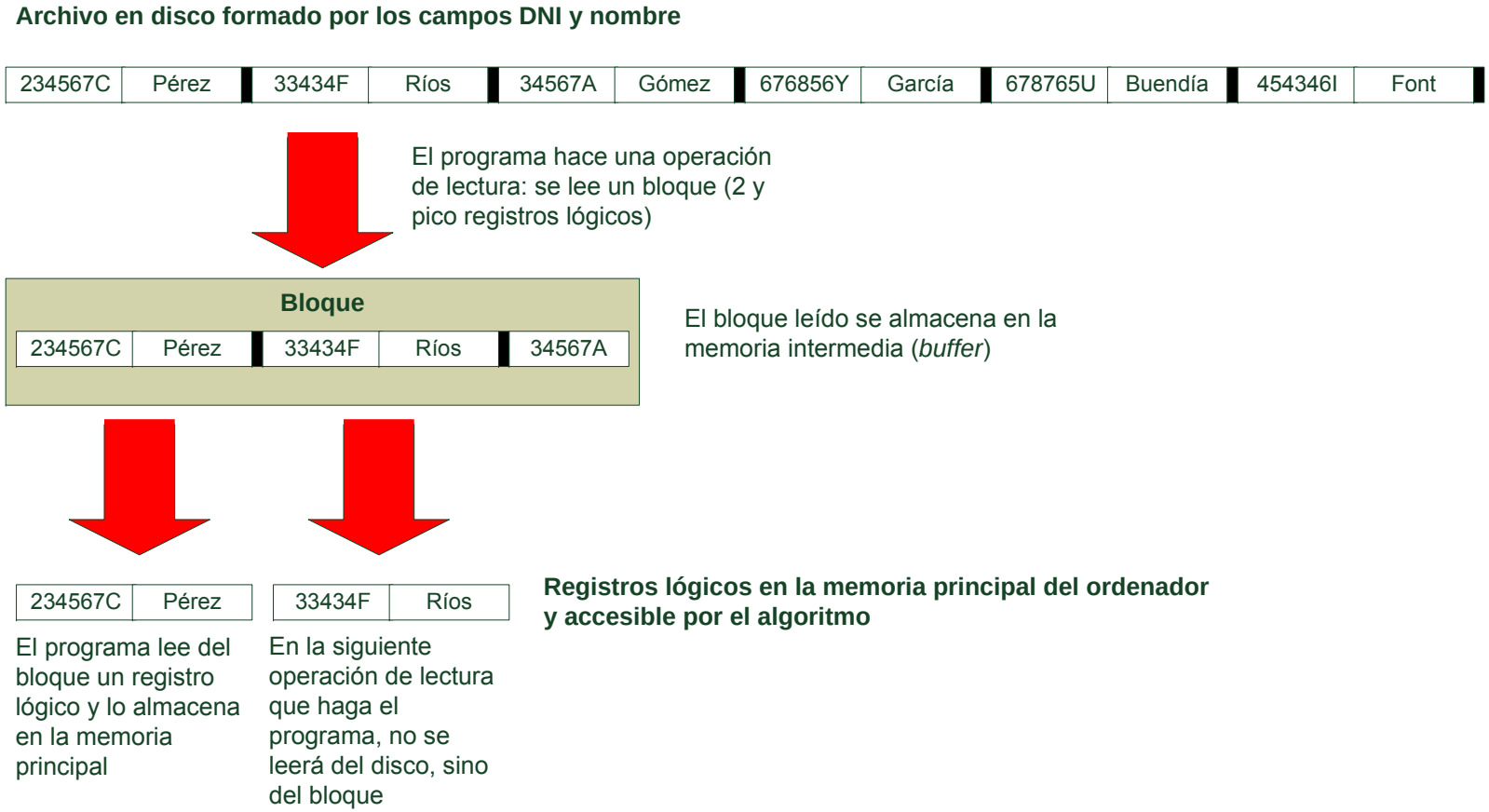
Estructura física

- La estructura física hará referencia a la forma en que el sistema operativo ve el archivo.
- Registro lógico (registro).
  - Unidad mínima de información a la que puede acceder un programa.
- Registro físico (bloque).
  - Unidad mínima de información que puede transferirse desde el soporte de almacenamiento a la memoria en una operación de entrada salida.
  - Desde el punto de vista físico, el archivo está compuesto por registros físicos o bloques.

Estructura física (II)

- Cuando el algoritmo solicita una operación de lectura o escritura en un archivo el sistema operativo no lee un registro lógico.
  - Se lee un bloque o registro físico que se almacena en un área de memoria intermedia (*buffer*).
    - ✓ Esa área de memoria será la zona de memoria identificada por la variable de tipo archivo.
  - El programa lee los registros lógicos desde el buffer.
    - ✓ Una vez almacenada la información en el buffer, se transfiere el registro lógico a la memoria del ordenador dónde puede acceder el algoritmo leyendo los registros lógicos.
    - ✓ Si se realiza otra operación de lectura, se busca la nueva información en el buffer.
      - ○ Si se encuentra, se realiza la transferencia al registro lógico.
      - ○ Si no se encuentra en el buffer, se realiza otra operación de lectura física.

Estructura física (III)



Estructura física (IV)

- Factor de bloqueo.
  - Mide la relación entre los registro físicos y lógicos.
    - ✓ Número de registros lógicos por bloque.
  - Tres casos:
    - ✓ Archivos bloqueados (factor de bloqueo > 1).
    - ✓ Archivos no bloqueados (factor de bloqueo = 1).
    - ✓ Archivos expandidos (factor de bloqueo < 1).
  - Bloqueo fuerte. Se aprovecha el espacio sobrante.
  - Bloqueo débil. El espacio sobrante no se aprovecha.

Estructura física (V)

- Importancia del factor de bloqueo.
  - El acceso a dispositivos externos es más costoso que el acceso a la memoria interna.
    - ✓ Al acceder a los registros lógicos a partir de la información almacenada en el buffer, se optimiza el acceso a la información.
  - En teoría un factor de bloqueo mayor acelerará los procesos de entrada/salida.
    - ✓ Con un factor de bloqueo grande cada operación de E/S cargará más información y, por lo tanto, habrá que acceder menos veces al disco.
      - ○ Los accesos a disco son más lentos.
    - ✓ Esto sólo es cierto cuando se desea acceder a toda la información de un archivo.
      - ○ Si se quiere acceder a los registros en orden aleatorio, un tamaño de bloque mayor también implicará un tiempo mayor para transferir la información a la memoria pero no ahorrará accesos a disco.

Soportes, tipos de acceso y organización de archivos

- La efectividad de un archivo a la hora de solucionar distintos problemas depende de cómo se haya organizado su información.
  - Cada tipo de organización permite algunos tipos de acceso.
  - Tanto la organización como el tipo de acceso depende del soporte (el tipo de dispositivo) dónde se han almacenado los datos.

Tipos de soportes

- Soporte:
  - Dispositivos de almacenamiento utilizados para guardar la información.
- Existen dos tipos de soportes:
  - Soportes secuenciales.
    - ✓ Almacenan cada registro uno a continuación de otros sin posibilidad de dejar espacios vacíos entre ellos.
    - ✓ Para acceder a un dato es necesario leer previamente los anteriores.
      - Para acceder al registro almacenado en la posición N habrá que leer los N-1 registros anteriores.
      - Ejemplos: cintas, tarjetas perforadas, impresoras, puertos de comunicaciones, etc.
    - ✓ La secuencia de acceso corresponde al orden físico de los registros.
  - Soportes direccionables.
    - ✓ Los mecanismos de lectura y escritura pueden situarse en cualquier momento en cualquier posición de su superficie:
      - Para acceder a un dato no es necesario recorrer los anteriores.
    - ✓ Es posible acceder directamente a un dato siempre que se conozca su posición (discos magnéticos, discos ópticos, etc.).
    - ✓ La secuencia de acceso no tiene por qué corresponder al orden físico de los registros.

Organización y tipo de acceso.

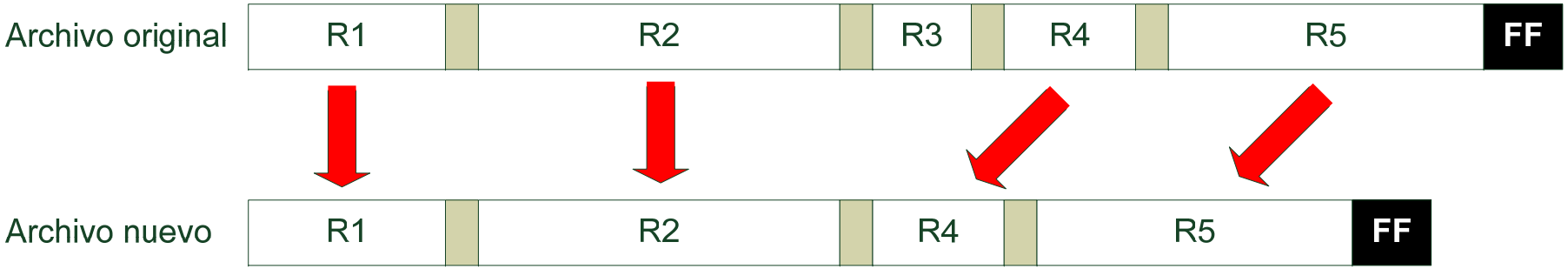
- Tipo de acceso.
  - Está ligado al tipo de soporte utilizado.
    - ✓ Acceso secuencial.
      - Implica recorrer todos los elementos del archivo uno a continuación del otro.
      - Dos posibilidades de movimiento: ir al siguiente elemento o ir al inicio del soporte.
      - Es el único posible en soportes secuenciales aunque se puede realizar también en soportes direccionables.
    - ✓ Acceso directo.
      - Acceder al elemento deseado sin necesidad de recorrer toda la información.
      - Si se conoce la posición del elemento y se dispone de un soporte direccionable es posible acceder al elemento deseado (los arrays permiten el acceso directo).
- Organización de un archivo.
  - Disposición interna de los registros en el soporte.
  - Forma de estructuración de los datos en el soporte.
    - ✓ Organización secuencial.
    - ✓ Organización directa o aleatoria.
    - ✓ Organización indexada o secuencial indexada.

Organización secuencial

- Los registros se disponen uno detrás de otro en el soporte sin dejar ningún espacio entre ellos.
- Los registros pueden ser de longitud variable.
  - Cada registro está separado por la marca de fin de registro.
- Al final del archivo existe una marca de fin de archivo.
- Sólo admiten un acceso secuencial.
  - En un momento dado sólo es visible un registro determinado y sólo se podrá ir al siguiente registro o posicionarse de nuevo en el primer registro.
- El orden físico de los registros corresponde a su orden lógico.
  - El acceso a la información se realiza por su orden físico.
  - La información se estructura (ordena) por algún tipo de secuencia lógica.
  - Cambiar el orden (insertar o eliminar) supone cambiar la disposición física de los registros.

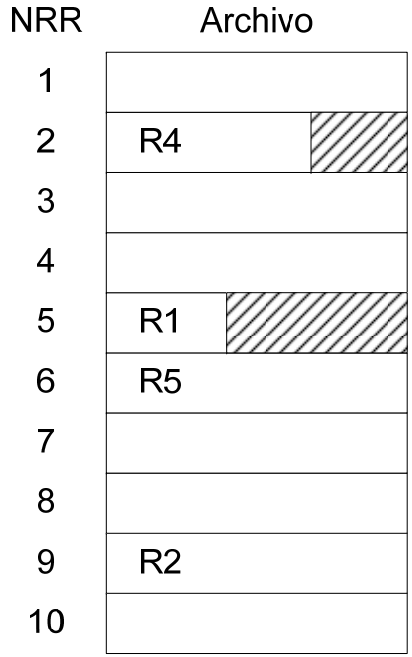
Organización secuencial (II)

- Cualquier modificación (insertar nuevos registros, modificar los datos, eliminar registros) exige la creación de un nuevo archivo con los datos modificados.
- Ejemplo: borrar el registro R3.



Organización directa

- Los registros se distribuyen de forma aleatoria por el soporte (archivos aleatorios).
  - Se almacenan en soportes direccionables.
- El archivo se divide en espacios (ranuras) de longitud fija que pueden estar libres u ocupadas total o parcialmente.
  - Los registros pueden ser de longitud variable
- A cada registro se accede por su posición relativa (archivos relativos).
  - Si se conoce la longitud de las ranuras y la posición de comienzo del archivo se podrá acceder a cualquier registro a partir de su número de registro relativo (NRR).
    - ✓ Archivos relativos en COBOL.



Organización directa (II)

- El orden lógico de acceso no se corresponde con el orden físico en que están almacenado.
  - Se podrán insertar nuevos registros en ranuras vacías, eliminar el contenido o modificar sus datos en cualquier momento.
- La relación entre la clave del registro y su posición la establece el programa que los maneje.
  - Generalmente esta relación se establecerá por alguna función de transformación de claves (*hashing*) que obtendrá la dirección de almacenamiento (NRR, número de registro relativo) a partir del campo clave del registro.

Organización directa (III)

- Necesitan un soporte direccionable.
- Admiten acceso secuencial y acceso directo.
  - Acceso directo.
    - ✓ Se averigua la posición relativa de un registro.
  - Acceso secuencial.
    - ✓ Se recorren todas las ranuras de forma correlativa.
    - ✓ Esto presenta un problema: ¿cómo distinguir las ranuras libres de las ocupadas?
      - ○ Algunos lenguajes como COBOL son capaces de distinguir si una ranura está libre u ocupada y saltarla al hacer una lectura secuencial o impedir la escritura si se intenta ocupar con otro registro.
    - ✓ El acceso secuencial no mantendrá el orden lógico de los registros.
      - ○ Suponiendo que el orden lógico de los registros es R1, R2, R4 y R5, el orden de lectura secuencial del archivo anterior sería R4, R1, R5 y R2.

Organización indexada

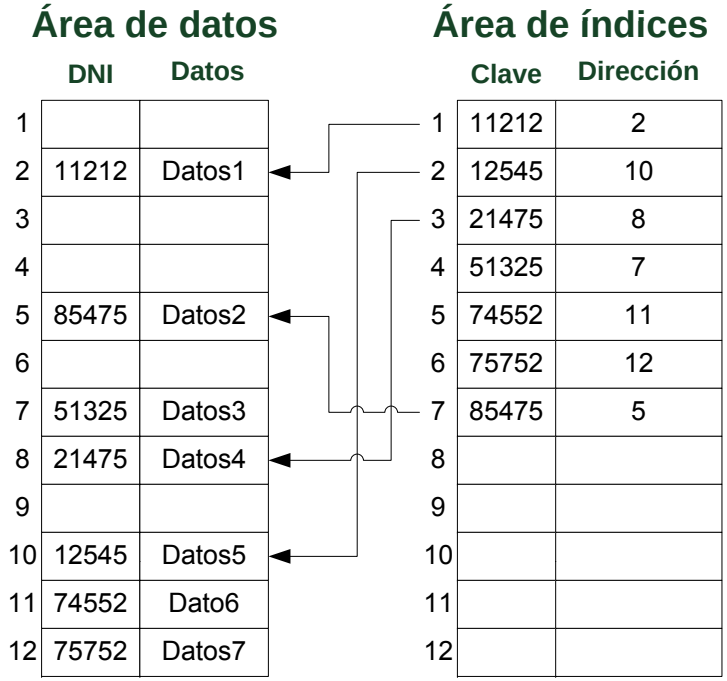
- El archivo está dividido en distintas áreas.
  - Área de datos.
    - ✓ Contiene la información normal del archivo.
    - ✓ Los registros se disponen en algún tipo de soporte que permita el acceso directo.
  - Área de índices.
    - ✓ Contiene una lista de las claves y la posición dónde está almacenada.
  - Área de excedentes.
    - ✓ Existe en algunas implementaciones.
    - ✓ Almacena los datos de los registros que no caben en el área principal.

Organización indexada (II)

- La relación entre la clave y su posición la establece el índice.
  - Cuando se almacena un nuevo registro en el área de datos, el índice debe reflejar su clave y su posición relativa.
    - ✓ La idea básica sería la del índice de un libro, donde aparecen las palabras que forman parte del índice (claves) y el número de página (posición del registro).
  - Pueden existir varios índices para permitir el acceso por varias claves.
    - ✓ Sólo es necesario crear índices para cada una de las claves por las que se quiere acceder.
      - Por ejemplo, en un libro puede haber un índice de lugares, de personas, etc.
- Existen distintas implementaciones:
  - Índices densos.
    - ✓ El índice contiene las claves de todos los registros del área de datos y su posición relativa.
    - ✓ En el área de índices los registros se ordenan por su clave.
  - Índices dispersos.
  - Índices multinivel.

Organización indexada (III)

- Acceso directo.
  - Se busca la clave en el índice, se obtiene la dirección donde está almacenada y se accede de forma directa al registro especificado en el área de datos.
- Acceso secuencial.
  - Se recorre secuencialmente el índice accediendo de forma correlativa y en secuencia de clave al área de datos.
- Inserción de registros.
  - Se inserta un elemento en el área de datos y a continuación, en secuencia de clave en el índice.
- Borrado de registros.
  - Se elimina la entrada del índice.



Ventajas e inconvenientes

- Organización secuencial.
  - Optimiza el espacio de almacenamiento.
  - Buena para acceder a todos los registros.
    - ✓ Por ejemplos en aplicaciones de listados, nóminas, etc., en las que sería la organización más eficiente.
  - No puede hacer un acceso directo.
- Organización directa.
  - Permite un acceso directo rápido a los registros.
    - ✓ Ideal si sólo se requiere realizar acceso directo y las claves admiten una transformación hash eficiente.
    - ✓ En esos casos la relación entre la clave de acceso y su posición es casi inmediata.
  - Dificultad en el acceso secuencial.
  - Ocupa más espacio que la organización secuencial ya que deja espacio entre registros.
- Organización indexada.
  - Permite un acceso secuencial y directo de una forma aceptable.
  - Permite acceder por varias claves.
  - El acceso secuencial siempre estará ordenado sin modificar el orden de los registros.
  - La organización secuencial realiza mejor los accesos secuenciales y la directa los directos.
  - El espacio de almacenamiento es mayor que en las otras organizaciones.

Instrucciones para archivos secuenciales

- Declaración de variables de tipo archivo secuencial

  archivo_s de *tipoDato* = varTipoArchivo

  - Reserva espacio en memoria para el buffer del archivo.
  - *tipoDato* es el tipo base del archivo puede ser cualquier tipo de dato estándar o definido por el usuario.
    - ✓ Archivos de texto: archivos secuenciales cuyo tipo base es un carácter.
  - En los archivos directos, se haría de la misma forma pero con la palabra reservada **archivo_d**.
  - El lenguaje algorítmico UPSAM no soporta archivos indexados de forma directa y hay que implementarlos utilizando archivos que permitan un acceso directo y otras estructuras de datos.

Instrucciones para archivos secuenciales (II)

- Apertura del archivo.
  - Antes de utilizar un archivo es necesario abrirlo
    abrir(*varArchivo, modo, nombreSistema*)
  - Abrir un archivo supone establecer una vía de comunicación (canal) entre el disco y la memoria principal, asignando un buffer al archivo en la variable de tipo archivo.
  - Antes de abrir un archivo es necesario que esté cerrado.
  - *varArchivo* es el nombre de la variable de tipo archivo que se haya declarado.
  - El *nombreSistema* es una cadena con la especificación de archivo.
    - ✓ Cada sistema operativo tiene puede especificar los archivos de forma distinta.
    - ✓ Por ejemplo en entornos Windows, podrían ser las cadenas:

```
'miArchivo.dat'
'datos\miArchivo.dat'
'c:\Mis documentos\datos\miArchivo.dat'
//Si la variable unidad contiene el valor 'D:\'
Unidad & 'miArchivo.dat'
```

- Creación del archivo.
  - Dependiendo del sistema y la organización, la creación de un archivo puede ir desde reservar espacio e inicializar el espacio de datos a simplemente crear un alta en el directorio de archivos.

Instrucciones para archivos secuenciales (III)

- Modos de apertura en archivos secuenciales.
  - **`lectura`**.
    - ✓ La única operación disponible es la lectura de registros.
    - ✓ Sitúa el puntero al siguiente registro (*next record pointer*, NRP) al comienzo del archivo.
    - ✓ El archivo debe existir previamente.
  - **`escritura`**.
    - ✓ Sólo se pueden hacer operaciones de escritura.
    - ✓ Sitúa el puntero al siguiente registro al comienzo del archivo, añadiendo registros.
      - En muchos lenguajes, el puntero al siguiente registro se sitúa al comienzo del archivo, sobrescribiendo los datos en el caso de que exista.
    - ✓ Si el archivo existe lo sobrescribe, en caso contrario, hace una llamada al sistema operativo para que cree el archivo en el sistema de archivos.
  - **`añadir`**.
    - ✓ Sólo se pueden hacer operaciones de escritura.
    - ✓ Sitúa el puntero después del último registro.
    - ✓ Si el archivo no existe, el sistema operativo lo crea.
  - Algunos lenguajes (COBOL) permiten abrir los archivos secuenciales en modo de **`lectura/escritura`**, colocando el NRP al comienzo del archivo y permitiendo operaciones de lectura y escritura.

Instrucciones para archivos secuenciales (IV)

- Cerrar el archivo.

  cerrar(*varArchivo1* [,*varArchivo2*…])

  - Rompe la vía de comunicación entre el archivo y el dispositivo.
  - Es necesario si el archivo se va a abrir en otro modo.
    - ✓ Puesto que sólo se puede abrir un archivo en un modo concreto, será necesario cerrarlo si se desea, por ejemplo, leer los datos después de escribirlos.
  - En los modos de escritura vuelca los datos del buffer al dispositivo.
    - ✓ Hay que recordar que los datos se escriben en el buffer, es decir, no se escriben en el disco sino en la memoria.
    - ✓ No cerrar el archivo puede suponer perder las modificaciones que se hayan hecho en memoria.
  - En los modos de escritura, coloca la marca de final de archivo después del último registro escrito.

Instrucciones para archivos secuenciales (V)

- Operaciones de lectura secuencial.

  leer(*varArchivo, variable*)

  - Lee el registro donde se encuentra el puntero y ésta pasa al siguiente registro.
  - Los datos leídos se cargan en la variable.
    - ✓ El tipo de dato de la información leída y de la variable deben coincidir.
      - Si la secuencia de bytes que se ha leído no se puede convertir al tipo de dato de la variable se genera un error.
  - Detecta error si se lee más allá de la marca de final de archivo.
- Operaciones de escritura secuencial.

  escribir(*varArchivo, expresión*)

  - Escribe el contenido de la expresión en el archivo especificado en la posición indicada por el puntero al siguiente registro.
  - El puntero se colocará al final del registro escrito.
  - La expresión será una variable en aquellos tipos de datos que no admitan constantes, expresiones aritméticas, etc.

Instrucciones para archivos secuenciales (VI)

```
var
   archivo_s de entero: núms
   entero: n
   cadena: c
inicio
   abrir(núms,escritura,'números.dat')

   escribir(núms,3)

   escribir(núms, 5+8)
   n ← 28
   escribir(núms, n)


   desde n ← 10 hasta 12 hacer
      escribir(núms, n)
   fin_desde



   cerrar(núms)
   ...
```

números.dat

3
13
28
10
11
12
FF

La orden abrir crea el archivo si no existe y sitúa el NRP al comienzo del archivo

Escribe el número 3 y el NRP pasa a la siguiente posición

Escribe un 13 en la posición actual y el NRP pasa a la siguiente posición

Escribe el valor de n en la posición actual y el NRP pasa a la siguiente posición

Escribe sucesivamente los valores de n

Al cerrar el archivo en un modo de escritura, se graba la marca de fin de archivo

Instrucciones para archivos secuenciales (VII)

```
var
   archivo_s de entero: núms
   entero: n,i
   cadena: c
inicio
   abrir(núms,lectura,'números.dat')

   leer(núms,n)

   leer(núms, n)

   desde 1 ← 1 hasta 4 hacer
      leer(núms, n)
   fin_desde

   leer(núms, n)

   cerrar(núms)
   ...
```

números.dat

3
13
28
10
11
12
FF

Abre el archivo para lectura y sitúa el NRP al comienzo del archivo

Lee el siguiente dato (un 3), lo intenta convertir a entero y lo almacena en la variable n y el puntero pasa al comienzo del siguiente registro

Asigna a n el valor 13 y pasa al siguiente registro

Escribe el valor de n en la posición actual y el NRP pasa a la siguiente posición

La variable n toma sucesivamente los valores 28, 10, 11, 12

Cómo ha llegado al final del fichero, no lee nada (en algunos lenguajes se generará un error)

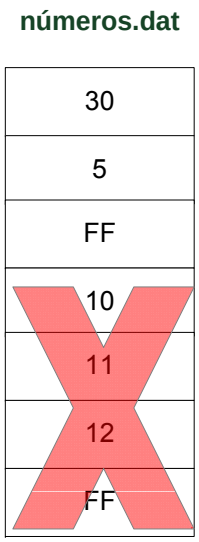
Instrucciones para archivos secuenciales (VIII)

```
var
   archivo_s de entero: núms
   entero: n
   cadena: c
inicio
   abrir(núms,escritura,'números.dat')

   escribir(núms,30)

   escribir(núms, 5)


   cerrar(núms)
```



La orden abrir crea el archivo si no existe y sitúa el NRP al comienzo del archivo

Escribe el número 30 sobre el primer registro del archivo y el puntero pasa al siguiente registro

Escribe un 5 sobre el segundo registro

Al cerrar el archivo en un modo de escritura, se graba la marca de fin de archivo.
Para el sistema operativo el archivo acabará aquí, tomando las posiciones siguientes como libres

Instrucciones para archivos secuenciales (VI)

- La marca de fin de archivo.
  - Al cerrar un archivo abierto en los modos de escritura se coloca la marca de fin de archivo que indicará al sistema operativo hasta dónde llega el mismo.
  - La función **`fda`**`(`*`varArchivo`*`)` devuelve un valor lógico verdadero si se ha detectado el final del archivo después de la última lectura.
  - Dos filosofías:
    - ✓ Pascal o C. La marca de final de archivo está asociada al último registro.
      - La función detecta el final del archivo cuando se lee el último registro.
    - ✓ COBOL. La marca de final de archivo está **después** del último registro.
      - Existe una especie de registro centinela situado después del último registro con la marca de final de archivo.
      - La función detecta el final de archivo cuando intentamos leer después del último registro.
- Otras operaciones con archivos.
  - **`borrar`**`(`*`nombreSistema`*`)`
  - **`renombrar`**`(`*`nombreSistemaAntiguo,nombreSistemaNuevo`*`)`

Operaciones con archivos secuenciales

- Recorrido.
  - Supone la creación de un bucle que procese todos los registros hasta detectar la marca de fin de archivo.
    - ✓ La marca de fin de archivo (EOF, FF, FDA) actúa como centinela, lo que implica una lectura anticipada.
  - Un recorrido típico seguirá las siguientes operaciones:

```
abrir el archivo para lectura
realizar una lectura anticipada
mientras no sea final del archivo hacer
   procesar el registro
   realizar una lectura secuencial
fin_mientras
cerrar el archivo
```

Operaciones con archivos secuenciales (II)

- Ejemplo de recorrido.
  - Suponemos un algoritmo que trabaja con un array de registros de productos con los campos código del producto, descripción y stock.
  - Se suponen las siguientes declaraciones:

```
const
   N = … //Número de productos a manejar
tipos
   registro = Producto
      cadena : código, descripción
      entero : stock
   fin_registro
   array[0..N] de Producto = Productos
```

Operaciones con archivos secuenciales (III)

- Cargar el array desde un archivo secuencial.
  - Si se tiene almacenado en el disco la información de los productos, se podría desarrollar un procedimiento que cargara los datos desde el archivo al array.

```
procedimiento CargarProductos(ref productos : p; ref entero : n)
//La variable N se cargará con el número total de registros leídos
var
   archivo_s de producto : A
   producto : R //El registro que se leerá del archivo
inicio
   abrir(A,lectura,'PRODUCTOS.DAT')
   leer(A,R)
   n ← 0
   mientras no fda(A) hacer
      n ← n + 1
      p[n] ← R
      leer(A,R)
   fin_mientras
   cerrar(A)
fin_procedimiento
```

Operaciones con archivos secuenciales (IV)

- Grabar el array en un archivo secuencial.
  - Al terminar el tratamiento del array, para no perder la información al terminar el programa, se debería grabar el contenido del array de nuevo en el archivo.

```
procedimiento GrabarProductos(valor productos : p; valor entero : n)
//La variable contiene el número de registros del array
var
   archivo_s de producto : A
   entero : i
inicio
   abrir(A,escritura,'PRODUCTOS.DAT')
   desde i ← 1 hasta n hacer
      escribir(A,p[i])
   fin_desde
   cerrar(A)
fin_procedimiento
```

Operaciones con archivos secuenciales
Consulta

- La consulta de un registro secuencial supone recorrer todos los anteriores. Es un tipo especial de recorrido.
  - Al procedimiento se le proporciona un registro con la clave que se está buscando y devuelve una variable lógica que indica si el registro se ha encontrado.
  - En la variable R también se almacenan los datos del registro que ha encontrado.

```
procedimiento Buscar(valor cadena:NombreArchivo; ref Producto :R;
                     ref lógico : encontrado)
var
   Producto : aux
   archivo_s de Producto : A
inicio
   abrir(A, lectura, NombreArchivo)
   leer(A,aux)
   mientras (aux.código <> R.código) y no fda(A) hacer
      leer(A, aux)
   fin_mientras
   encontrado ← aux.código = producto.código
   si encontrado entonces
      R ← aux
   fin_si
   cerrar(A)
fin_procedimiento
```

Operaciones con archivos secuenciales
Bajas

- Borrar un registro en un archivo secuencial supone copiar a un archivo auxiliar todos los registros menos el que se quiera borrar.
  - El procedimiento recibe un registro con el valor de la clave a borrar y devuelve una variable lógica que indica si se ha producido la operación de borrado.
  - Al final del procedimiento se elimina el archivo original (con datos no actualizados) y se renombra el archivo auxiliar con el nombre original ya que es el que contiene los datos actualizados.

```
procedimiento Borrar(valor cadena:NombreArchivo; valor producto:R; ref lógico : borrado)
var
   producto : RAux
   archivo_s de producto : A, AAux
inicio
   abrir(A,lectura,NombreArchivo)
   abrir(AAux,escritura,'AUX.DAT')
   leer(A,RAux)
   borrado ← falso
   mientras no fda(A) hacer
      si R.código = RAux.código entonces
         borrado ← verdad
      si_no
         escribir(AAux,RAux)
      fin_si
      leer(A, RAux)
   fin_mientras
   cerrar(A,AAux)
   borrar(NomArch)
   renombrar('AUX.DAT', NombreArchivo)
fin_procedimiento
```

Operaciones con archivos secuenciales
Modificaciones

- Para modificar un registro en un archivo secuencial se deben copiar en un archivo auxiliar todos los registros.
  - El registro que se desee modificar se guardará con sus datos modificados.
  - Al procedimiento se le pasan un registro con la clave del registro a modificar y los nuevos datos del mismo.

```
procedimiento Modificar(valor cadena:NombreArchivo; valor producto:R;ref lógico : modificado)
var
   procuto : RAux
   archivo_s de producto : A, AAux
inicio
   abrir(A,lectura,NombreArchivo)
   abrir(Aaux, escritura,'AUX.DAT')
   leer(A,RAux)
   modificado ← falso
   mientras no fda(A) hacer
      si R.código = Raux.cóidgo entonces
         modificado ← verdad
         escribir(AAux,R)
      si_no
         escribir(AAux,RAux)
      fin_si
      leer(A, RAux)
   fin_mientras
   cerrar(A,AAux)
   borrar(NomArch)
   renombrar('AUX.DAT', NombreArchivo)
fin_procedimiento
```

Operaciones con archivos secuenciales
Añadir

- Añadir un registro al final.
  - El modo de apertura añadir, abre el archivo y mueve el puntero al siguiente registro después del último registro.
  - Cualquier operación de lectura añadirá los nuevos datos al final del archivo.

```
procedimiento Añadir(valor cadena:NombreArchivo; valor producto:R)
var
   archivo_s de  producto : A
inicio
   abrir(A,añadir,producto)
   escribir(A,R)
   cerrar(A)
fin_procedimiento
```

Operaciones con archivos secuenciales
Insertar

- Insertar un registro manteniendo la estructura lógica.
  - Hay que (1) escribir todos los registros con clave menor que el registro a insertar, (2) escribir el registro y (3) Copiar el resto de datos.

```
procedimiento Insertar(valor cadena:NomArch;valor producto : R)
var
   archivo_s de  producto : A,AAux
   producto : RAux
inicio
   abrir(A,lectura,NomArch)
   abrir(Aaux, escritura,'AUX.DAT')
   leer(A,RAux)
   //Copiar todos los registros menores que R a AAux
   mientras (R.código > RAux.código) y no fda(A) hacer
      escribir(AAux,RAux)
      leer(A,RAux)
   fin_mientras
   escribir(AAux,R)
   //Copiar todos los registros que quedan
   mientras no fda(A) hacer
      escribir(AAux,RAux)
      leer(A,RAux)
   fin_mientras
   cerrar(A,AAux)
   borrar('AUX.DAT')
   renombrar('AUX.DAT',NomArch)
fin_procedimiento
```

Ejercicios

1. En un archivo secuencial se tienen almacenadas las notas de una serie de alumnos (el número máximo posible de alumnos es de 200). Por cada alumno se almacena su expediente, el código de la asignatura y la nota obtenida en el último examen.
   - Declare las estructuras de datos necesarias para almacenar la información.
   - Diseñe un subprograma que permita cargar a los alumnos almacenados en el archivo en un array de registros con los mismos campos. El subprograma también deberá devolver el número total de alumnos que se han cargado en el array.
   - Diseñe un subprograma que permita almacenar los alumnos del array de registros en un archivo secuencial.

Ejercicios (II)

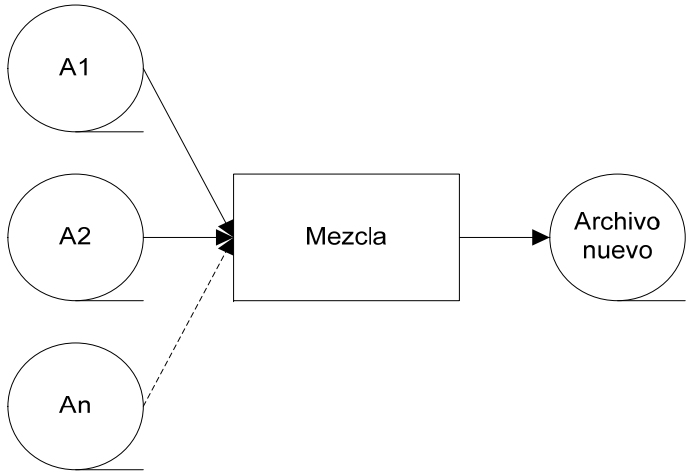
2. Una empresa tiene almacenado en un archivo secuencial las ventas realizadas por sus vendedores (la empresa tiene 25 vendedores) a lo largo de la semana.
   - Por cada registro del archivo se guarda:
     - ✓ Número del vendedor (un número del 1 al 25).
     - ✓ Nombre del vendedor.
     - ✓ Día de la semana (un número del 1 al 7).
     - ✓ Cantidad vendida.
   - Se desea:
     - ✓ Almacenar las ventas en una tabla de 25 x 7 elementos, de forma que en la posición i,j de la tabla se almacenen las ventas del vendedor i en el día j.
     - ✓ Almacenar los nombres de los vendedores en un array de registros. Cada registro del array tendrá los campos nombre y total ventas. En la posición i del array se almacenarán los datos del vendedor número i.
     - ✓ Calcular el total de las ventas de cada vendedor.
     - ✓ Almacenar el array de registros en un archivo secuencial.

Ejercicios (III)

3. Una empresa de trabajo temporal mantiene a sus empleados en un array de registros desordenado con los campos DNI, nombre del empleado, sueldo y fecha del final del contrato (en formato aaaammdd).
   - Codificar un subprograma que aumente el sueldo a un empleado cuyo DNI se pasará como argumento. El subprograma modificará el archivo secuencial y el sueldo se incrementará en un tanto por ciento que también se pasará como argumento al subprograma.
   - Codificar un subprograma que elimine del archivo a todos los empleados cuya fecha de final de contrato sea mayor o igual a otra fecha que se pasará como argumento.

Operaciones con archivos secuenciales
Mezcla

- Obtener un nuevo archivo de salida a partir de dos o más archivos de entrada de forma que el nuevo archivo mantenga la misma estructura lógica que los archivos de entrada.



Archivo A1

Clave	Datos
10	
23	
45	
51	
63	
88	
96	
Final del archivo	

Archivo A2

Clave	Datos
12	
18	
31	
47	
49	
105	
125	
Final del archivo	

Archivo nuevo

Clave	Datos
10	
12	
18	
23	
31	
45	
47	
49	
51	
63	
88	
96	
105	
125	
Final del archivo	

Operaciones con archivos secuenciales
Mezcla (II)

- Es necesario realizar una lectura sincronizada de los dos archivos.
  - Una vez se tiene en memoria un registro de cada archivo de entrada:
    - ✓ Se selecciona aquel que tenga la clave más pequeña.
    - ✓ Se lleva al archivo de salida.
    - ✓ Se lee el siguiente registro del archivo que se haya seleccionado.

Operaciones con archivos secuenciales
Mezcla (III)

```
algoritmo mezcla
tipos
   registro = tipoReg
      entero : clave
      //Resto de datos
   fin registro
   archivo_s de tipoReg = tipoArch
var
   tipoArch : A,A1,A2
   tipoReg : R1,R2
inicio
   abrir(A,escritura,'NUEVO.DAT')
   abrir(A1,lectura,'ARCHIVO1.DAT')
   abrir(A2,lectura,'ARCHIVO2.DAT')
   leer(A1,R1)
   leer(A2,R2)
   mientras no fda(A1) o no fda(A2) hacer
      si r1.clave < r2.clave entonces
         escribir(A,R1)
         leer(A1,R1)
      si_no
         escribir(A,R2)
         leer(A2,R2)
      fin_si
   fin_mientras
   cerrar(A,A1,A2)
fin
```

Operaciones con archivos secuenciales
Mezcla (III)

- El algoritmo anterior puede presentar un problema.
  - En un momento determinado se tienen en memoria el registro con clave 92 de A1 y el registro con clave 105 de A2.
  - Como 92 es menor se selecciona ese registro, se lleva al archivo de salida y se lee el siguiente registro de A1.
  - Como es final de archivo, la operación de lectura no se realiza, por lo que en memoria sigue estando el registro con clave 92.
  - Al volverlo a comparar con el registro del archivo A2, se volvería a seleccionar sin acabar nunca el proceso.
- Aunque el problema tiene varias soluciones, una de ellas sería llevar el valor máximo posible a la clave del registro cuando se llegue al final de fichero.
  - El lenguaje COBOL permite hacer eso en la operación de lectura:

    ```
    READ A1 AT END MOVE HIGH-VALUE TO clave
    ```

  - Se puede simular este mecanismo añadiendo unas líneas de código al algoritmo anterior.

Operaciones con archivos secuenciales
Mezcla (V)

```
...
  leer(A1,R1)
  leer(A2,R2)
  mientras no fda(A1) o no fda(A2) hacer
     si r1.clave < r2.clave entonces
        escribir(A,R1)
        leer(A1,R1)
     si_no
        escribir(A,R2)
        leer(A1,R2)
     fin_si
     si fda(A1) entonces
         R1.clave ← +∞ //Se mueve el valor mayor posible a la clave
     fin_si
     si fda(A2) entonces
         R2.clave ← +∞ //Se mueve el valor mayor posible a la clave
     fin_si
  fin_mientras
  ...
```

- De esta forma, al leer el último registro se movería el mayor valor posible a la clave de R1 y todos los registros siguientes de A2 serían menores y se volcarían en el archivo nuevo.

Operaciones con archivos secuenciales
Actualización por lotes

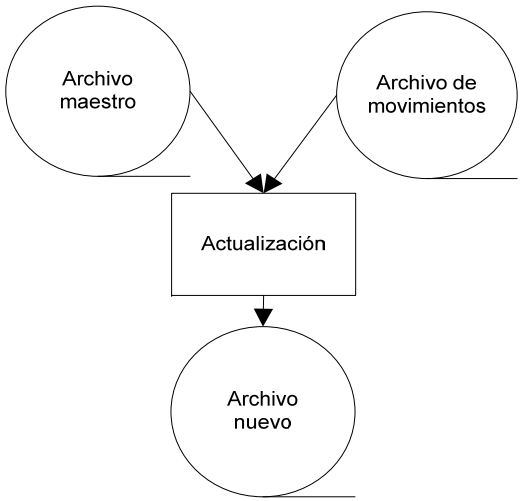
- Procesos interactivos y procesos batch.
  - Procesos interactivos.
    - ✓ Precisan la presencia de un operador.
    - ✓ Útiles para accesos directos y procesar pocos registros.
    - ✓ No son útiles cuando se utilizan archivos secuenciales.
  - Procesos por lotes, en diferido o batch.
    - ✓ No precisan la presencia de un operador.
    - ✓ Son los adecuados cuando se utiliza un acceso secuencial.
    - ✓ La salida y mensajes de usuario se deben hacer por algún tipo de dispositivo como impresoras u otros archivos.

Operaciones con archivos secuenciales
Actualización por lotes (II)

- El enfrentamiento de archivos es una variación de la mezcla.
  - Normalmente se realizará una actualización:
    - ✓ Se modifica el archivo de salida en virtud del valor de los campos.
    - ✓ Se trabaja, al menos, con tres archivos:
      - Archivo maestro.
        - Archivo que contiene todos los datos originales (sin actualizar).
      - Archivo de movimientos.
        - Archivo con las modificaciones que se harán sobre los datos originales.
      - Archivo de salida.
        - Archivo de salida que contiene los datos originales modificados con la información del archivo de movimientos.
    - ✓ Todos los archivos deben estar ordenados por la misma clave.
  - A partir de los datos de entrada (archivo maestro y archivo de movimientos) se consigue un archivo actualizado.

Operaciones con archivos secuenciales
Actualización por lotes (III)

- Los dos archivos se leen de forma sincronizada.
  - Si la clave (el DNI) es igual, se mueve el nuevo sueldo al registro de salida.
  - En caso contrario el registro queda como está.
  - Se escribe el registro (modificado o no) en el archivo de salida.



Archivo Maestro

DNI	Nom	Dep	Suel
1111	Nombre1	CONTABILIDAD	1000
2222	Nombre2	CONTABILIDAD	1200
3333	Nombre3	DESARROLLO	1300
4444	Nombre4	VENTAS	1000
5555	Nombre5	CONTABILIDAD	1100
8888	Nombre8	VENTAS	1300

Archivo de Movimientos

DNI	Suel
1111	1200
3333	1500
4444	1100
8888	1400

Archivo nuevo de salida

DNI	Nom	Dep	Suel
1111	Nombre1	CONTABILIDAD	1200
2222	Nombre2	CONTABILIDAD	1200
3333	Nombre3	DESARROLLO	1500
4444	Nombre4	VENTAS	1100
5555	Nombre5	CONTABILIDAD	1100
8888	Nombre8	VENTAS	1400

Operaciones con archivos secuenciales
Actualización por lotes (IV)

```
tipos
  …
  registro = RMovimiento
     cadena: dni
     real : suel
  fin_registro
  archivo de RMovimiento : AMovimientos
  registro = REmpleado
     cadena: dni,nom,dep
     real : suel
  fin_registro
  archivo de REmpleado : Aempleado
   …
var
  AEmpleado: A,ANue
  AMovimientos: AMov
  REmpleado: R
  RMovimiento : RMov
```

```
inicio
  abrir(A,lectura,'EMPLEADOS.DAT')
  abrir(AMov,lectura,'MOVIM.DAT')
  abrir(ANue,escritura,'NUEVO.DAT')
  leer(A,R)
  leer(AMov,RMov)
  mientras no fda(A) hacer
      si R.dni = RMov.dni entonces
          R.suel ← Rmov.suel
          leer(AMov,RMov)
      fin_si
      escribir(ANue,R)
      leer(A,R)
   fin_mientras
   cerrar(A,AMov,ANue)
fin
```

Operaciones con archivos secuenciales Actualización por lotes (V)

- En otras ocasiones, en lugar de modificar simplemente un campo, la actualización puede llevar a cabo todas las operaciones de mantenimiento del archivo: altas, bajas y modificaciones.
  - En estos casos el archivo de movimientos tendrá todos los campos del maestro y estará cargado con la información a dar de alta, a modificar o la clave de los registros a borrar.
  - Tendrá además un campo más para indicar el tipo de movimiento del archivo.
    - ✓ Podría ser A para un alta, B para una baja o M para una modificación.
  - El proceso sería el siguiente:
    - ✓ Si un registro del maestro no tiene movimientos (la clave del maestro es menor que la de movimientos), el registro se lleva tal y como está al archivo de salida.
    - ✓ Si un registro de archivo de movimientos no existe en el maestro (la clave de movimientos es menor que la del maestro), se trata de un alta y se insertará el registro de movimientos en el archivo de salida.
    - ✓ Si el registro del maestro tiene movimientos (las clave son iguales) puede ser una baja o una modificación.
      - ○ Si es una baja (tipo de movimiento B) no se escribe en el archivo de salida.
      - ○ Si es una modificación se pasan los campos de movimientos al registro de salida y se graba.

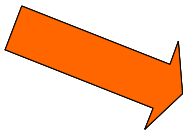
Operaciones con archivos secuenciales
Actualización por lotes (VI)

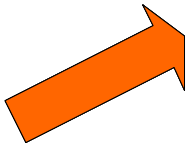
ARCHIVO MAESTRO

DNI	Nom	Dep	Suel
2222	Nombre2	CONTABILIDAD	1200
3333	Nombre3	DESARROLLO	1300
4444	Nombre4	VENTAS	1000
8888	Nombre8	VENTAS	1300
9999	Nombre9	VENTAS	1200

ARCHIVO MOVIMIENTOS

DNI	Nom	Dep	Suel	Tipo
1111	Nombre1	VENTAS	1200	A
3333	Nombre3	DESARROLLO	1400	M
4444				B
5555	Nombre5	DESARROLLO	1000	A
8888	Nombre8	DESARROLLO	1400	M





ARCHIVO NUEVO

DNI	Nom	Dep	Suel
1111	Nombre1	VENTAS	1200
2222	Nombre2	CONTABILIDAD	1200
3333	Nombre3	DESARROLLO	1000
5555	Nombre5	DESAROLLO	1000
8888	Nombre8	DESARROLLO	1400
9999	Nombre9	VENTAS	1200

Operaciones con archivos secuenciales.
Actualización por lotes (VII)

```
algoritmo Actualización
tipos
   //Definición del registro del archivo maestro y del nuevo
   registro = REmpleado
      cadena: dni, nombre, dep
      real : suel
   fin_registro
   archivo_s de REmpleado = AEmpleado
   registro = RMovimiento
      cadena: dni,nom,dep
      real : suel
      carácter : tipo
   fin_registro
   archivo de RMovimiento : AMovimientos
var
   AEmpleado: A,ANue
   AMovimientos: AMov
   REmpleado: R,RNue
   RMovimiento : Rmov
inicio
   abrir(A,lectura,'EMPLEADOS.DAT')
   abrir(AMov,lectura,'MOVIM.DAT')
   abrir(ANue,escritura,'NUEVO.DAT')
```

Operaciones con archivos secuenciales.
Actualización por lotes (VIII)

```
leer(A,R)
leer(AMov,RMov)
mientras no fda(A) o no fda(AMov) hacer
  si RMov.dni < R.dni entonces
     //RMov no existe en el maestro. Es un alta o un error
     //Primero se mueven los campos del registro de movimiento
     RNue.DNI ← RMov.DNI
     RNue.nombre ← RMov.Nombre
     RNue.dep ← RMov.dep
     RNue.suel ← RMov.suel
     escribir(ANue,RMov)
     leer(AMov,RMov)
  si_no
     si R.dni < RMov.dni entonces
        //R no tiene movimientos se graba el registro tal cual
        escribir(ANue,R)
     si_no
        //RMov existe en el maestro. Es una baja o modificación, o un error
        si RMov.tipo = 'M'
          //Primero se mueven los campos del registro de movimiento
           RNue.DNI ← RMov.DNI
           RNue.nombre ← RMov.Nombre
           RNue.dep ← RMov.dep
           RNue.suel ← RMov.suel
```

Operaciones con archivos secuenciales.
Actualización por lotes (IX)

```
            escribir(ANue,RMov)
          fin_si
          leer(AMov,Rmov)
        fin_si
        leer(A,R)
    fin_si
    //Si se llega al final de los archivos se mueve el mayor valor a la clave
    si fda(AMov) entonces
       RMov.DNI ← +∞
    fin_si
    si fda(A) entonces
       R.DNI ← +∞
    fin_si
  fin_mientras
  cerrar(A,AMov,ANue)
fin
```

Operaciones con archivos secuenciales
Rupturas de control

- Agrupar registros dependiendo del valor de uno o más campos.
- Se suele utilizan para realizar acumulaciones parciales.
- Existirán uno o más campos de ruptura.
  - Campo de ruptura: campo por el que se va a agrupar.
  - El archivo debe estar ordenado por ese campo.
- Se irán acumulando los valores mientras el campo de ruptura no cambie y no sea final de fichero.
- Ejemplo: Sumar los sueldos de los empleados por departamento.

Archivo (ordenado por el campo Dep)

DNI	Nom	Dep	Suel
1111	Nombre1	CONTABILIDAD	1000
2222	Nombre2	CONTABILIDAD	1200
5555	Nombre5	CONTABILIDAD	1100
3333	Nombre3	DESARROLLO	1300
4444	Nombre4	VENTAS	1000
8888	Nombre8	VENTAS	1300

Listado

Departamento	Total sueldos departamento
CONTABILIDAD	3300
DESARROLLO	1300
VENTAS	2300

Operaciones con archivos secuenciales. Rupturas de control (II)

```
algoritmo Ruptura
tipos
   //Definición del registro del archivo
   registro = REmpleado
      cadena: dni, nombre, dep
      real : suel
   fin_registro
   archivo_s de REmpleado = AEmpleado
var
   AEmpleado: A
   REmpleado: R
   cadena: depAux //Necesario para controlar el cambio de departamento
   real : TotalDep //Necesario para acumular los sueldos
inicio
   abrir(A,lectura,'EMPLEADOS.DAT')
   leer(A,R)
   mientras no fda(A) hacer
      //Por cada departamento se inicializa el total
      TotalDep ← 0
      //Por cada departamento se inicializa la variable auxiliar
      depAux ← R.dep
      mientras (R.dep = depAux) y no fda(A) hacer
          TotalDep ← TotalDep + R.suel
          leer(A,R)
      fin_mientras
      escribir(depAux, TotalDep)
   fin_mientras
   cerrar(A)
fin
```

Operaciones con archivos secuenciales
Ordenación de archivos

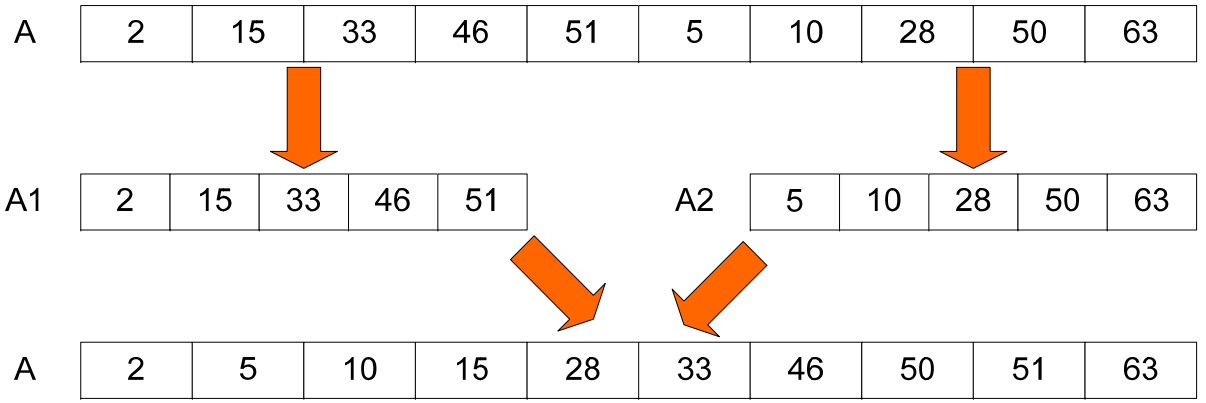
- En ocasiones se puede ordenar un archivo utilizando un array de forma auxiliar.
  - Volcar archivo en el array.
  - Ordenar el array.
  - Volcar el array de nuevo en el archivo.
- Pero este método no es aplicable si el volumen de datos a ordenar es demasiado grande y no es posible volcar todos los datos del archivo en un array en memoria, o no tenemos forma de averiguar el número aproximado de elementos del archivo.
- En esos casos se deben utilizar métodos de ordenación distintos a los que se utilizan en arrays.
  - En un array tenemos todos los datos al mismo tiempo en memoria y es posible acceder en un momento determinado a cualquier dato del array, cambiar su valor o moverlo de posición.
  - En un archivo secuencial, en un momento dado sólo podemos acceder a un dato y no podemos modificarlo o cambiarlo de posición.
- Los métodos de ordenación de archivos se basan en crear particiones ordenadas del archivo y mezclar esas particiones.

Operaciones con archivos secuenciales
Ordenación de archivos (II)

- Partición de un archivo.
  - Consiste en distribuir los datos de un archivo en dos o más archivos.
    - ✓ La partición se puede hacer:
      - En virtud del valor de un campo (por ejemplo, los usuarios de Madrid van a un archivo, los del resto de comunidades a otro).
      - En secuencias de un número fijo de registros (por ejemplo, los n primeros registros van al archivo A1, los n siguientes al archivo A2, los n siguientes al archivo A1, los siguientes al archivo A2, etc.).
      - En secuencias ordenadas...
    - ✓ En general, cuando se hacen particiones de un archivo para ordenar, se tratará de dividir el archivo en secuencias ordenadas distribuidas en uno o más archivos de salida.
- Mezcla de un archivo.
  - Se vio anteriormente el procedimiento de mezcla.
  - Si un archivo se parte en dos secuencias ordenadas y se mezclan esas secuencias, el resultado será un archivo ordenado.

Operaciones con archivos secuenciales
Ordenación de archivos (III)

- Supongamos que tenemos el archivo A que está formado por dos secuencias ordenadas de registros:
  - Por una parte 2, 15, 33, 46, 51, y por otra 5, 10, 28, 50, 63

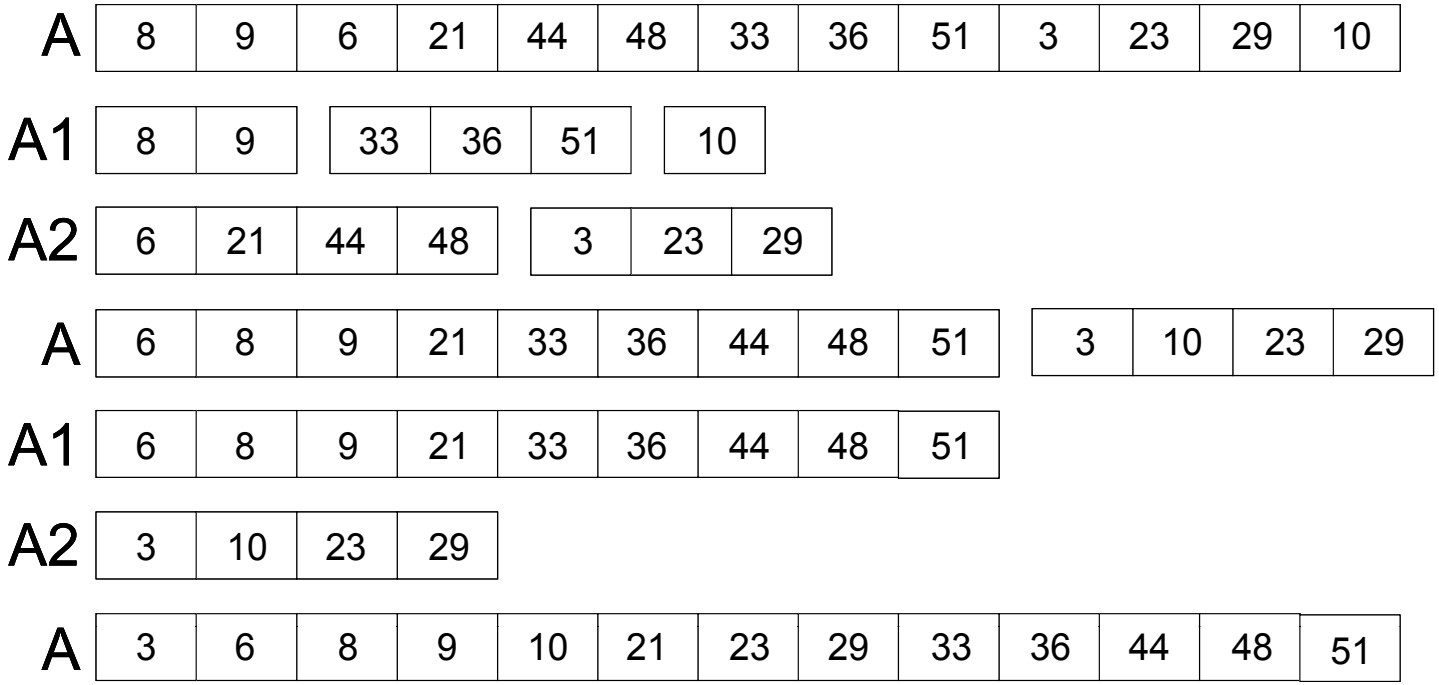


  - Si dividimos el archivo en dos mitades ordenadas (A1 y A2) y las mezclamos según el procedimiento de mezcla ya visto, tendríamos el archivo ordenado.
- Esta situación ideal (un archivo con dos mitades ordenadas) no se dará habitualmente, pero...
  - Esta es la base de la ordenación de archivos y en algún momento del proceso de ordenación llegará a darse esa situación.

Operaciones con archivos secuenciales
Ordenación de archivos (IV)

- Método de mezcla natural.
  - Realiza dos particiones (o más) del archivo pero no totalmente ordenadas, sino compuestas de varias secuencias ordenadas.
  - Al mezclar dos de esas secuencias tendremos secuencias ordenadas de tamaño mayor.
  - Si se repite el proceso con las secuencias ordenadas de las dos particiones tendremos un archivo con secuencias ordenadas de mayor longitud (con la mitad de secuencias).
  - Se vuelven a realizar dos particiones con secuencias ordenadas y se repite al proceso hasta que cada partición tiene sólo una secuencia ordenada.

Operaciones con archivos secuenciales
Ordenación de archivos (V)



Operaciones con archivos secuenciales
Ordenación de archivos (VI)

```
algoritmo Ordenar
tipos
   registro = tipoReg
      entero : clave
      //Resto de datos
   fin_registro
   archivo_s de tipoReg = tipoArch
var
   entero : n //n es el número de secuencias mezcladas en el procedimiento mezcla
inicio
   repetir
      //Parte el archivo ARCHIVO.DAT en dos particiones AUX1 y AUX2
      //Cada partición estará formada por las secuencias ordenadas
      //de ARCHIVO.DAT
      partición('ARCHIVO.DAT','AUX1','AUX2')
      //Mezcla las secuencias ordenadas de AUX1 y AUX2 en el
      //archivo ARCHIVO.DAT. La variable n indica el número
      //de secuencias que se han mezclado
      mezcla('AUX1','AUX2','ARCHIVO.DAT',n)
   //Cuando el número de secuencias mezcladas es 1
   //el archivo está ordenado
   hasta_que n = 1
fin
```

Operaciones con archivos secuenciales
Ordenación de archivos (VII)

```
procedimiento partición(valor cadena : NomArch,NomArch1,NomArch2)
var
   entero : i
   lógico : GrabarA1
   tipoReg : R,RAux
   tipoArch : A,A1,A2
inicio
   abrir(A1,escritura,NomArch1)
   abrir(A2,escritura,NomArch2)
   abrir(A,lectura,NomArch)
   GrabarA1 ← verdad
   leer(A,R)
   mientras no fda(A) hacer
      si GrabarA1 entonces
          escribir(A1,R)
      si_no
          escribir(A2,R)
      fin_si
      RAux ← R
      leer(A,R)
      si R.clave < RAux.clave entonces
          GrabarA1 ← no GrabarA1
      fin_si
   fin_mientras
   cerrar(A)
   cerrar(A1)
   cerrar(A2)
fin_procedimiento
```

Operaciones con archivos secuenciales
Ordenación de archivos (VIII)

```
procedimiento mezcla(valor cadena:NomArch1,NomArch2,NomArch;ref entero: n)
var
   tipoArch : A,A1,A2
   tipoReg : R1,R2,RAux1,RAux2
inicio
   abrir(A,escritura,NomArch)
   abrir(A1,lectura,NomArch1)
   abrir(A2,lectura,NomArch2)
   leer(A1,R1)
   leer(A2,R2)
   n ← 0
   mientras no fda(A1) o no fda(A2) hacer
      RAux1 ← R2
      RAux2 ← R2
      mientras no fda(A1) y no fda(A2) y (R1.clave>=RAux1.clave)
                                       (R2.clave >=RAux2.clave) hacer
          si r1.clave < r2.clave entonces
              escribir(A,R1)
              RAux1 ← R1
              leer(A1,R1)
```

Operaciones con archivos secuenciales
Ordenación de archivos (IX)

```
            si_no
              escribir(A,R2)
              RAux2 ← R2
              leer(A2,R2)
            fin_si
        fin_mientras
        mientras no fda(A1) y (R1.clave >= Raux1.clave) hacer
            escribir(A,R1)
            RAux1 ← R1
            leer(A1,R1)
        fin_mientras
        mientras no fda(A2) y (R2.clave >= RAux2.clave) hacer
            escribir(A,R2)
            RAux2 ← R2
            leer(A2,R2)
        fin_mientras
        n ← n + 1
    fin_mientras
    cerrar(A,A1,A2)
fin_procedimiento
```

Archivos de texto

- Caso especial de archivos secuenciales.
  - Archivo cuyo tipo base son caracteres.
  - Declaración:

    ```
    archivo_s de carácter : varArchivo
    ```

  - Detección del final de línea.
    - ✓ Función **`fdl`**`(varArchivo)`.
      - Devuelve un valor verdadero si el último carácter leído es el de final de línea (ASCII 13).

Archivos de texto (II)

- Ejemplo: detectar el número de instrucciones en un archivo de C.

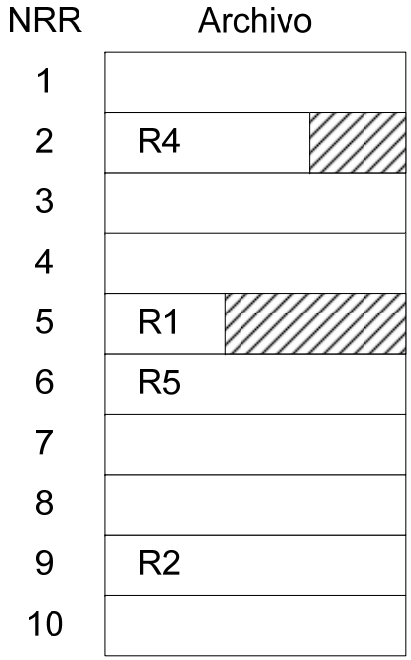
```
algoritmo ContarInstrucciones
//Si cada instrucción va seguida de punto y coma,
//si se cuentan los caracteres ; se sabrá el número de instrucciones
var
   archivo_s de carácter : programa
   carácter : c
   entero : conta
inicio
   abrir(programa,lectura,'EJEMPLO.C')
   leer(programa,c)
   conta ← 0
   mientras no fda(programa) hacer
      si c = ';' entonces
         conta ← conta + 1
      fin si
      leer(programa,c)
   fin_mientras
   escribir(conta)
   cerrar(programa)
fin
```

Ejercicios

4. Un Ayuntamiento tiene almacenada las multas que impone a los vehículos de su municipio en un archivo secuencial (MULTAS.DAT). Por cada multa, se almacena un identificador único por cada denuncia, la matrícula del vehículo, el DNI del conductor y la fecha de la denuncia (en formato aaaammdd) y está ordenado por el identificador de la denuncia. Mensualmente se genera otro archivo secuencial con las denuncias realizadas (MULTAS_MES.DAT) con los mismos campos y ordenado por la fecha de la denuncia.
   - Diseñar un programa que permita añadir el archivo MULTAS_MES.DAT al archivo MULTAS.DAT. Tenga en cuenta que ambos archivos están ordenados por campos distintos. Si es necesario puede realizar la ordenación utilizando un array auxiliar.
5. Periódicamente, también se genera un archivo con los recursos admitidos a las denuncias (RECURSOS.DAT). En dicho archivo se almacena únicamente el identificador de la denuncia. Diseñar un programa que permita eliminar del archivo MULTAS.DAT todos los recursos admitidos. Tenga en cuenta, que por error puede que haya recursos que ya hayan sido borrados previamente. En esos casos aparecerá un mensaje advirtiendo del error.
   - Diseñar un programa que permita eliminar todos los recursos admitidos del archivo MULTAS.DAT.
6. En un archivo secuencial se guarda información sobre una serie de productos. Por cada producto se almacena su código y el stock disponible en el almacén. Codifique un algoritmo que guarde en otro archivo secuencial aquellos productos con stock negativo, borrándolos además del archivo original.

Archivos de organización directa
Conceptos

- Archivos formados por “huecos” de longitud fija que pueden estar o no ocupados por información.
- El orden lógico no se corresponde con el orden físico.
- El acceso se puede hacer de forma secuencial o directa.
  - En el acceso directo se accede por la posición relativa del registro, por su *número de registro relativo* (NRR).



Trabajo con archivos directos

- Declaración de variables de tipo archivo directo

  archivo_d de *tipoDato* = varTipoArchivo

  - Reserva espacio en memoria para el buffer del archivo.
  - El tipo base del archivo puede ser cualquier tipo de dato estándar o definido por el usuario.
- Creación del archivo.
  - Dependiendo del sistema y la organización, la creación de un archivo puede ir desde reservar espacio e inicializar el espacio de datos a simplemente crear un alta en el directorio de archivos.

Trabajo con archivos directos (II)

- Apertura del archivo.

  abrir(*varArchivo, modo, nombreSistema*)

- Modos de apertura en archivos directos.
  - **lectura**.
  - **escritura**.
  - **lectura/escritura**.
    - ✓ El archivo debe existir.
    - ✓ Es el modo habitual de trabajar con este tipo de archivos.
    - ✓ Sitúa el puntero al siguiente registro al comienzo del archivo, permitiendo leer o escribir registros.

Trabajo con archivos directos (III)

- Cerrar el archivo.

  cerrar(*varArchivo1* [,*varArchivo2*…])

  - Rompe la vía de comunicación entre el archivo y el dispositivo.
  - Es necesario si el archivo se va a abrir en otro modo.
  - En los modos de escritura vuelca los datos del buffer al dispositivo.
    - ✓ No cerrar el archivo puede suponer perder las modificaciones que se hayan hecho en memoria.
  - En los archivos directos no es necesario abrir y cerrar en cada operación que se haga:
    - ✓ Al poderse abrir en modo de lectura y escritura, no es necesario cerrar para cambiar el tipo de operación que se haría en el archivo.
    - ✓ Normalmente se abre el archivo al comienzo del programa y se cierra al final de su ejecución.

Trabajo con archivos directos (IV)

- Operaciones de lectura.
  - Lectura secuencial.

    leer(*varArchivo, variable*)

    ✓ Lee el registro donde se encuentra el puntero y ésta pasa al siguiente registro.

    ✓ La lectura secuencial permite utilizar la función **fda** para detectar el final del archivo.
  - Lectura directa.

    leer(*varArchivo, posición, variable*)

    ✓ Lee el registro situado en número de registro relativo *posición* y mueve el puntero al siguiente registro.

    ✓ No detecta ningún error si intentamos leer más allá del último registro actual del archivo.

Trabajo con archivos directos (V)

- Operaciones de escritura.
  - Escritura secuencial.

    escribir(*varArchivo, expresión*)

    ✓ Escribe el contenido de la expresión en el archivo especificado en la posición indicada por el puntero al siguiente registro, sobreescribiendo en el caso de estar ocupada la posición.
  - Escritura directa.

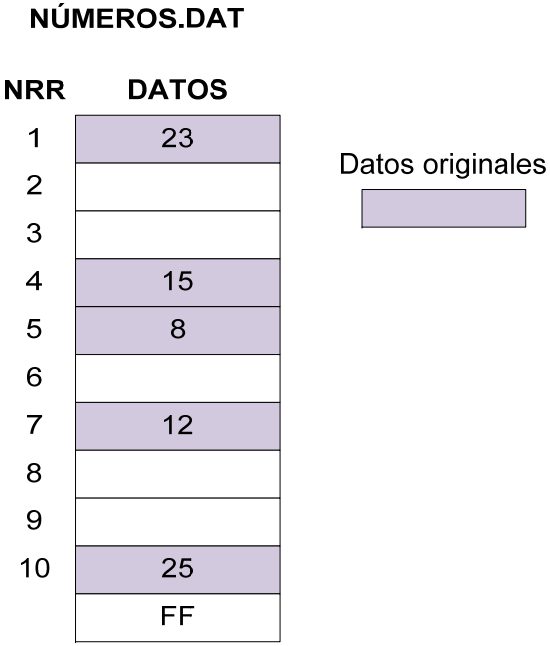
    escribir(*varArchivo, posición, expresión*)

    ✓ Escribe el contenido de la expresión en el archivo especificado en la posición indicada, sobreescribiendo en el caso de estar ocupada la posición.

    ✓ Si se escribe más allá del último registro, se amplía el archivo hasta dicha posición.

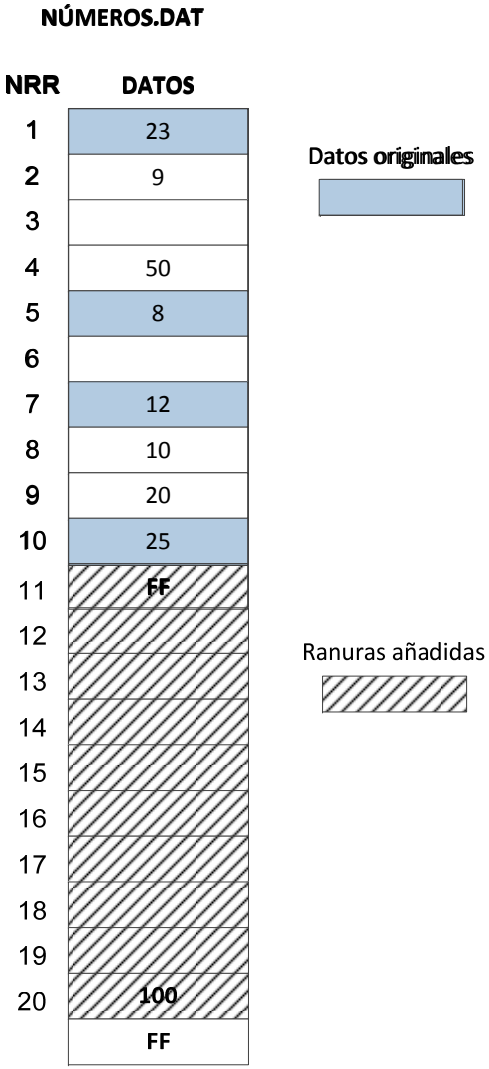
Trabajo con archivos directos (VI)

```
var
   archivo_d de entero: números
   entero: posición, núm
inicio
   //Abre el archivo y coloca el puntero en el primer dato
   abrir(números,lectura/escritura, 'NUMEROS.DAT')
   //Lectura secuencial, núm vale 23
   leer(números,núm)
   //Lectura directa del registro 7. Mueve el puntero a la
   //posición 7. Lee el 12 y el puntero pasa al siguiente dato
   leer(números, 7,núm)
   //Mueve el puntero a la posición 4, lee el 15 y el puntero
   //pasa al siguiente dato
   leer(números,4,núm)
   //Lectura secuencial. Lee el siguiente dato (el 8)
   leer(números, núm)
   //Teóricamente se pude leer más allá del último registro.
   //En ese caso leería lo que estuviera en el soporte en esa
   //posición 1000
   leer(números, 1000, núm)
   //Lee el registro 10 y se pasa a la siguiente posición
   leer(números, 10, núm)
   //Lectura secuencial. Como es fin de archivo, no lee y núm
   // sigue valiendo 25
   leer(números,núm)
```



Trabajo con archivos directos (VII)

```
   //Mueve el puntero a la posición 2 y escribe un 9
   //en esa posición. El puntero pasa a la siguiente posición
   posición ← 2
   escribir(números, posición, 9)
   //Mueve el puntero a la posición 8 y escribe un 10
   // en esa posición. El puntero pasa a la siguiente posición
   posición ← 8
   escribir(números, posición, 10)
   //Escritura secuencial. Escribe un 20 en la siguiente
   //posición
   escribir(números, 20)
   //Escribe un 50 en la posición 4. Como está ocupada por otro
   // número sobrescribe el dato que había (un 15)
   posición ← 4
   escribir(números, posición, 50)
   //Escribe el número 100 en la posición 20 del archivo
   //Ese será el nuevo último registro
   //Todas las ranuras entre la posición 10 y la 20 forman ahora
   //parte del archivo
   escribir(números,20,100)
fin
```



Trabajo con archivos directos (VIII)

- Detección del estado de un registro.
  - Los archivos directos pueden sobrescribir sobre posiciones ya ocupadas o leer sobre posiciones vacías.
    - ✓ Lenguajes como COBOL pueden detectar si la posición está libre u ocupada, impidiendo la lectura en una posición libre o la escritura en una posición ocupada.
    - ✓ Si el lenguaje no tiene esa capacidad (consideraremos que el lenguaje UPSAM 2.1 no la tiene), es necesario un mecanismo que detecte si una posición está libre u ocupada.
      - Clave 0 para registros vacíos o distintos de 0 para los ocupados.
      - Un valor lógico a verdad o falso en un campo adicional según esté o no ocupada.
      - Un valor de tipo carácter o entero en un campo adicional que detecte el estado del registro (por ejemplo, 0 libre y 1 ocupado, o 'L' libre y 'O' ocupado.
    - ✓ Es necesario inicializar como libres todas las posiciones antes de utilizar el archivo:
      - Determinar el número total de registros que va a contener el archivo.
        - Se considera la existencia de una constante `MaxReg` con el número total del registros.
      - Realizar un programa que permita crear (inicializar) todo el espacio.

Archivos con transformación de claves

- El problema de los archivos directos para encontrar un registro a partir de un campo clave es que la posición del registro no guarda ninguna relación con la clave
- Para trabajar con archivos directos es necesario transformar una clave en una dirección del almacenamiento válida (número de registro relativo, NRR)
- Para la transformación se aplica una función de transformación de clave (función *hash*) a una clave para que devuelva una posición de almacenamiento válida.

  `dirección ← hash(clave)`

  - La función hash devolvería un número entre 1 y n, siendo n el número de posiciones del archivo.
  - Todos los elementos del archivo deberían almacenarse utilizando la misma función hash.
- El método de transformación de claves ideal es *`dirección ← clave`*.
  - Sólo es posible si las claves son correlativas.
  - Desaprovecha espacio de almacenamiento si las claves no son correlativas.
    - ✓ Por ejemplo, si queremos almacenar 100 personas y la clave es el DNI (la persona con DNI 45.432.543 ocuparía la posición relativa 45.432.543 del archivo).
    - ✓ Para almacenar a cada persona en la posición que diga su DNI necesitaríamos reservar aproximadamente 60.000.000 de posiciones (una por cada DNI posible), desaprovechando 59.999.900 posiciones.
    - ✓ Una función hash trataría de convertir el DNI en una dirección válida de almacenamiento, por ejemplo, en un número entre 1 y 100.

Métodos de transformación de claves

- Cometidos:
  - Transformar una clave numérica o alfanumérica en un dirección de almacenamiento válida.
- Deben esparcir los elementos por el espacio lo máximo posible.
  - La dispersión aumenta si aumenta el número de posiciones de almacenamiento (siempre que no aumente el número de claves).
    - ✓ La relación entre el número de posiciones reservadas y el número de registros reales se llama **factor de carga**. (número de registros / número máximo de registros).
  - Además del espacio necesario para almacenar los datos, se reservará espacio adicional, es decir un factor de carga mayor que 1.
    - ✓ Una medida estándar es dejar aproximadamente aproximadamente de un 20% de espacio libre, es decir un factor de carga de 0,8.
- Algunas funciones hash:
  - Aritmética modular
  - Truncamiento.
  - Plegamiento.
  - Mitad del cuadrado.

Métodos de transformación de claves (II)

- Aritmética modular (módulo de la división entera).

  `hash(clave)` → `clave` **mod** `n + 1`

  - Devuelve un valor entre 1 y `n`.
  - `n` es un número menor o igual que el número de posiciones a distribuir.
    - ✓ Funciona mejor si `n` es un número impar o, mejor aún, primo.

Clave	hash(clave)
1203	40
6754	62
32	33

Para 100 registros, n podría ser 97

hash(clave) → (clave **mod** 97) + 1

	Clave	Resto de campos
33	32	
40	1203	
62	6754	

Métodos de transformación de claves (III)

- Truncamiento.
  - Extraer n dígitos de la parte central de la clave.
  - n será el número de dígitos máximo de la posición.
  - Se utiliza con claves numéricas grandes.

```
entero función hash(valor entero : clave)
//Para una dirección relativa de 3 dígitos
//Extrae las centenas de millar, decenas de millar y millares
var
   entero : dirección
inicio
   dirección ← clave mod 1000000
   devolver(dirección div 1000 + 1)
fin_función
```

  - Para la clave 53454654, para 1000 posiciones, devolvería...

clave mod 1000000 div 1000 + 1 = 455.

Métodos de transformación de claves (IV)

- Plegamiento.
  - Partir la clave en fragmentos de `n` dígitos y sumarlos, despreciando el acarreo.
  - `n` será el número de dígitos del número máximo de posiciones.

```
entero función hash(E entero : clave)
//Para una dirección relativa de 3 dígitos
var
   entero : suma,resto
inicio
   suma ← 0
   resto ← clave
   mientras resto > 0 hacer
     suma ← suma + resto mod 1000
     resto ← resto div 1000
   fin_mientras
   devolver(suma mod 1000 + 1)
fin_función
```

  - Para la clave 53454654 para 1000 posiciones, devolvería

$$(654+454+53) \bmod 1000 + 1 = 162$$

Métodos de transformación de claves (V)

- Mitad del cuadrado.
  - Elevar la clave al cuadrado y aplicarle el truncamiento.
- Para claves no numéricas.
  - Convertir la clave alfanumérica en un dígito.
    - ✓ Cambiando cada carácter por su valor según el código ASCII.
    - ✓ Sumando el código ASCII de cada carácter.
  - Aplicar alguno de los métodos anteriores.

```
entero :  función hash(E cadena :clave)
//Para un archivo con 101 direcciones aplica la aritmética modular
var
   entero : suma, i
inicio
   suma ← 0
   //La función longitud devuelve el número de caracteres de una cadena
   //La función código devuelve el código asociado a un carácter (por ejemplo, el carácter A devuelve 65
   desde i ← 1 hasta longitud(clave) hacer
      suma ← suma + código(clave[i])
   fin_desde
   devolver(suma mod 101 + 1)
fin_función
```

- Para la clave 'AABACB' para 100 posiciones devolvería:

65+65+66+65+67+66 mod 100 + 1 = 95

Tratamiento de colisiones

- Todas estas funciones presentan un problema:
  - A no ser que se utilice la función hash `hash(`*`clave`*`)` → *`clave`* puede que claves distintas generen la misma dirección.
  - Dos claves que generen la misma dirección son **sinónimos**.
  - Cuando se trata de almacenar dos sinónimos se produce una **colisión**.
- Cualquier método de transformación de claves produce direcciones iguales para claves distintas: **sinónimos**.
- Cuando se dan sinónimos se produce una **colisión**: dos claves de intentan almacenar en la misma posición.
- Es necesario tratar las colisiones para evitar sobreescribir registros.

Tratamiento de colisiones (II)

- Dos problemas:
  - Será necesario averiguar si una posición de almacenamiento ya está ocupada por otra clave.
    - ✓ Se puede inicializar alguno de los campos o la propia clave a un valor centinela (por ejemplo -1).
  - Será necesario buscar un nuevo espacio en el almacenamiento para guardar el sinónimo.
    - ✓ Distintas técnicas.
      - Encadenamiento de colisiones.
      - Llevar las colisiones a una zona espacial (direccionamiento a zona de colisiones).
      - Almacenar los sinónimos lo más cerca posible de su posición original (direccionamiento a vacío).
- Resolución de colisiones:
  - Área de colisiones.
  - Direccionamiento a vacío.
  - *Buckets* (reservar varias posiciones para los sinónimos).
  - Encadenamiento de colisiones.

Tratamiento de colisiones
Direccionamiento a vacío

Clave	hash(clave)
1203	4
6754	35
32	33
8683	44
839	120
1363	44
1079	120

hash(clave) → clave mod 120 + 1

NRR	Clave	Resto de datos
1	1079	
	-1	
4	1203	
	-1	
33	32	
	-1	
35	6754	
	-1	
44	8683	
45	1363	
	-1	
	-1	
	-1	
120	839	

Tratamiento de colisiones
Direccionamiento a vacío (II)

- Inicialmente todos las posiciones están vacías (campo clave = -1).
- Se aplica la función hash a la clave 1203 (el resultado es 4).
  - Se accede a la posición 4 y cómo está vacía se almacenan ahí los datos del registro con clave 1203.
- El mismo proceso se aplica a las claves 6754, 32, 8683 y 839.
- Se aplica la función hash a la clave 1363 (el resultado es 44).
  - Cómo está ocupada (la clave es distinta de -1) se accede a la posición siguiente.
  - Cómo la posición 45 está libre se almacenan ahí los datos del registro con clave 1363.
  - Si hubiera estado ocupada se iría pasando a la siguiente posición hasta encontrar una posición libre.
    - ✓ Siempre habrá una posición libre puesto que tenemos 100 registros posibles y hemos reservado 120 posiciones.
- Se aplica la función hash a la clave 1079 (el resultado es 120).
  - Cómo esta ocupada se accede a la posición siguiente.
  - Al ser la última posición del archivo, se considerará que la posición siguiente es la 1 (simularíamos un "archivo circular").
  - Cómo la posición 1 está libre se almacenan ahí los datos.
  - Si hubiera estado ocupada si accedería a las posiciones 2, 3, 4, ... hasta encontrar una libre.

Archivos de organización directa
Ejemplo

- Se desea gestionar un archivo de almacén. Se prevé que se va a tener un máximo de 100 productos distintos, y por cada uno de ellos se almacenará el código de producto, una descripción del producto y el stock.
  - Se ha añadido un campo más (estado) para indicar si el archivo está libre (0), ocupado (1) o dado de baja (2).
  - Para un manejo más eficiente de los elementos se reservarán 120 posiciones para almacenar los 100 productos.
  - En cada una de esas 120 posiciones, se marcará inicialmente el campo estado con un valor 0 para los registros libres (que inicialmente serán todos).
  - El archivo Tema02-Archivos_Gestion_de_un_archivo_directo_de_productos.pdf disponible en el campus virtual contiene el código del ejemplo.

Organización indexada

- Contienen, al menos, dos áreas:
  - Área de datos.
    - ✓ De acceso directo.
    - ✓ Contiene todos los datos del archivo.
    - ✓ Los registros pueden estar o no ordenados en secuencia de clave.
  - Área de índices.
    - ✓ Contiene una clave y una dirección (NRR) de dónde se almacena.
  - Puede tener, en algunos casos, un área de excedentes.
    - ✓ Necesaria si los registros están ordenados en el área de datos.
- La relación entre la clave y su posición la da el índice.
  - En los secuenciales la da el orden físico.
  - En los directos la relación la proporciona la transformación de claves.
  - En los archivos indexados una vez que se han almacenado los datos la clave y la posición que ocupa en el área de datos deben quedar relacionadas por el índice.
- Si tenemos varios índices se podría acceder por varias claves.

Organización indexada
Índices densos

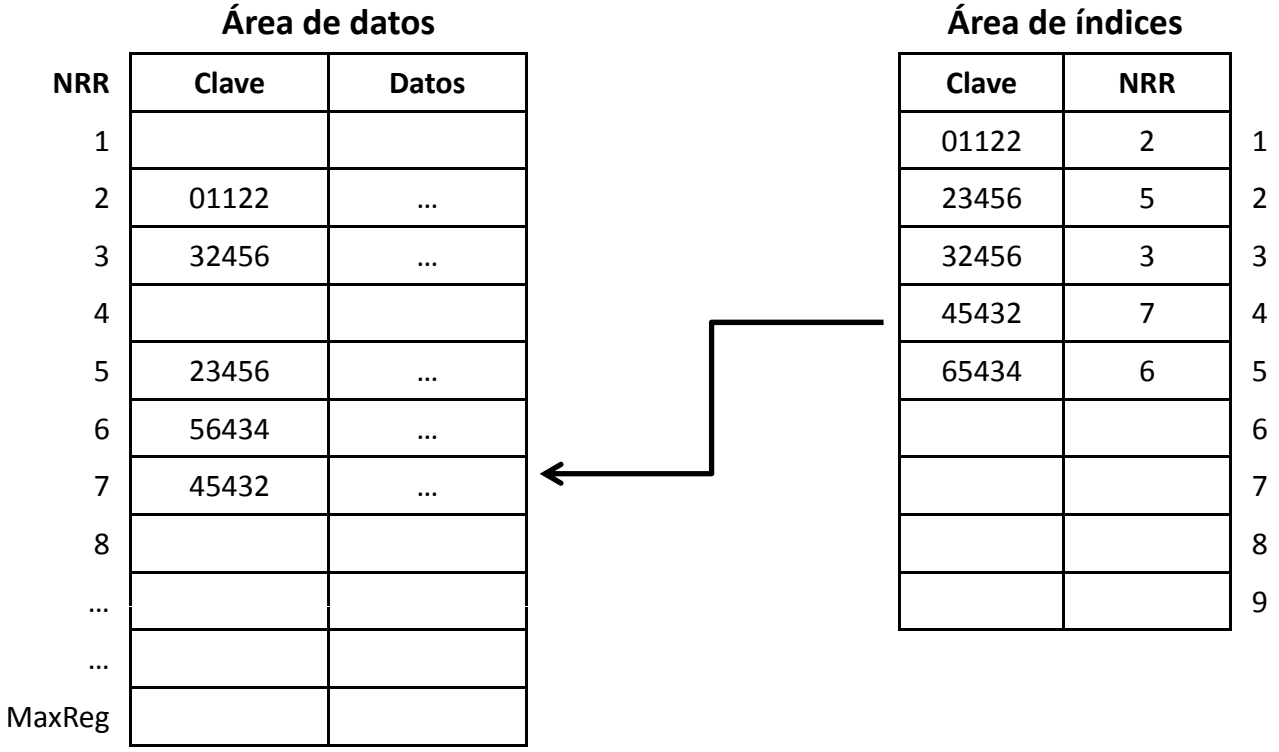
- Existe una entrada del índice por cada registro en el área de datos.
  - Cada entrada contiene la clave por la que se accederá a la información y la posición de los datos en el área de datos.
- El área de datos deberá se de acceso directo.
- El área de índices está ordenada por la clave.
- Acceso directo.
  - Se realiza un búsqueda de la clave en el área de índices.
    - ✓ Si las claves están almacenadas en un array, se podría hacer una búsqueda binaria.
  - Una vez localizada la clave, se obtiene su posición en el área de datos (su número de registro relativo) y con ella se accede de forma directa a la información.
- Acceso secuencial.
  - También se hace por el área de índices.
  - Se recorre secuencialmente el área de índices.
  - Por cada elemento del índice al que se accede se obtiene su posición en el área de datos y se accede directamente a la información.

Organización indexada
Índices densos (II)

- Insertar un registro.
  - Se busca una posición libre en el área de datos.
  - Se crea una nueva entrada en el índice, insertando de forma ordenada la clave y la posición dónde se ha almacenado.
- Modificación.
  - Se realiza un acceso secuencial y se modifican los datos a partir del NRR obtenido en el índice.
- Eliminación.
  - Se realiza un acceso secuencial y se elimina la entrada en el área de índices.
  - Para optimizar el espacio en el área de datos, se podrá marcar ese dato como borrado (baja lógica).

Organización indexada
Índices densos (III)

Buscar la clave 45432



Organización indexada
Índices dispersos

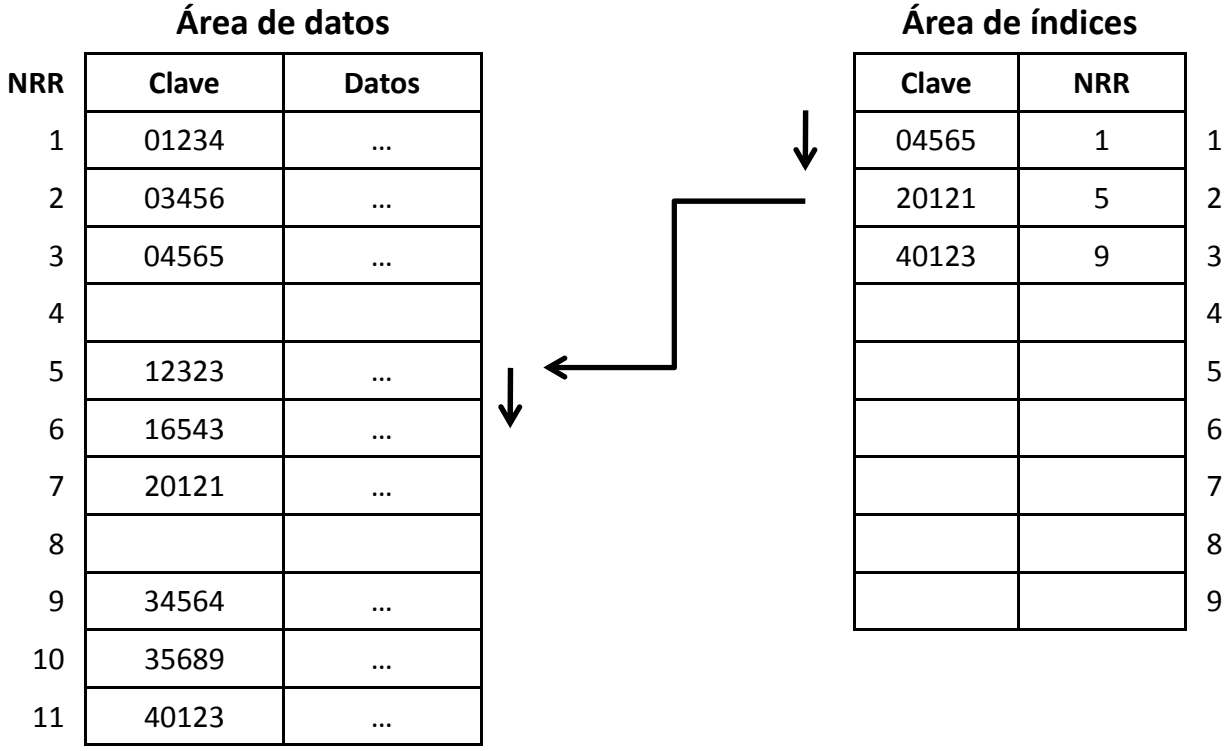
- El problema de los índices densos es que si el número de claves es muy grande pueden llegar a ocupar mucho espacio.
- En un índice disperso (organización secuencial indexada) el área de índices **no contiene** todas las claves.
- El área de datos contiene la información ordenada y agrupada por bloques.
- El área de índices almacena una entrada por cada bloque.
  - La entrada contiene la clave mayor del bloque y el valor de la clave mayor del bloque.
- Acceso directo.
  - Combina los índices con un acceso secuencial.
    - ✓ Busca la clave secuencialmente en el índice hasta encontrar una clave mayor o igual que la que se está buscando.
      - Localiza el bloque dónde podría estar el dato.
    - ✓ A partir de ahí, obtiene la posición de inicio del bloque y accede al primer registro del bloque en el área de datos.
    - ✓ Recorres secuencialmente el bloque hasta encontrar una clave mayor o igual que la que se está buscando.
      - Si es mayor, el registro no está.

Organización indexada
Índices dispersos (II)

- Acceso secuencial.
  - Al estar los datos ordenados, accede secuencialmente a la información en el área de datos.
- Insertar un nuevo registro.
  - Primero se intenta añadir el registro dentro del bloque dónde le correspondería estar.
    - ✓ Para ello, entre bloque y bloque se pueden dejar posiciones libre.
    - ✓ Si hay sitio disponible (hay espacio entre bloques), se inserta el nuevo dato manteniendo el bloque ordenado.
    - ✓ La estructura del índice no cambiaría.
  - Si no hay sitio para insertarlo en su bloque correspondiente se almacenaría en un área de excedentes y se relacionaría el nuevo dato con su bloque correspondiente.

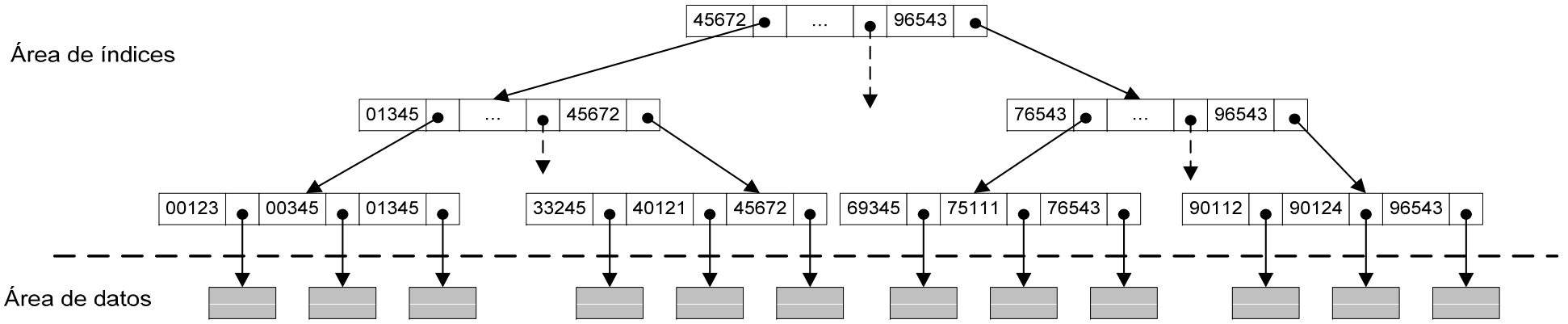
Organización indexada
Índices dispersos (III)

Buscar la clave 16543



Organización indexada (IV)

- Índices multinivel.
  - Se estructuran en forma de árbol.
    - ✓ En el primer nivel se encuentran las clave ordenadas de menor a mayor y la dirección del siguiente nivel del árbol.
    - ✓ En el nivel inferior se encuentran las claves ordenadas y punteros hacia el área de datos.
  - Esta es la estructura del sistema de archivo ISAM o VSAM de IBM.



Organización indexada: ejemplos.

- Ejemplo 1.
  - Se desea indexar el archivo directo del ejemplo anterior mediante un índice denso.
    - ✓ Al final habrá que almacenar en un archivo secuencial los índices.
- Ejemplo 2.
  - Gestión del archivo de almacén creado en el ejemplo anterior.
    - ✓ Procedimientos para cargar el índice al comienzo del proceso y guardarlo al final del programa.
    - ✓ Altas de un nuevo registro.
    - ✓ Baja de un registro.
    - ✓ Modificaciones.
    - ✓ Acceso directo a partir de la clave.
    - ✓ Acceso secuencial.
- El archivo Tema02-Archivos_Gestion_de_un_archivo_indexado_de_productos.pdf disponible en el campus virtual contiene el código de los ejemplos.

Ejercicios

7. Diseñe un programa que genere un archivo directo con transformación de claves a partir de los datos del archivo MULTAS.DAT de la dispositiva 72. El archivo tendrá capacidad para 10.000 denuncias y la clave primaria será el identificador de la denuncia.
8. Diseñe un programa que genere un archivo indexado a partir de los datos del archivo MULTAS.DAT (diapositiva 72). Se supone que el archivo tendrá una capacidad de 10.000 denuncias y la clave primaria del archivo será el identificador de la denuncia.

Ejercicios (II)

9. Se tiene un archivo de personal indexado con los campos DNI, nombre y sueldo. El archivo tiene previsto almacenar un máximo de 100 empleados. El índice se almacena temporalmente en un archivo secuencial que contiene la clave (el DNI) de cada empleado y la posición que ocupa dentro del área de datos.

   Se dispone también de un archivo secuencial con los pluses que tienen algunos empleados. Cada registro de este archivo tendrá los campos DNI y plus.

   Se desea actualizar el archivo de personal incrementando el sueldo en la cantidad que indique el plus de cada empleado.

   Por distintos errores, en este archivo pueden existir DNI que no estén presentes en el archivo indexado. En ese caso se borrarán del archivo secuencial.

10. Se desea actualizar el archivo secuencial que contiene los pluses del ejercicio anterior con otro archivo también secuencial y de la misma estructura que almacena los nuevos pluses que dará la empresa a sus empleados.

    En el archivo actualizado, se añadirán los pluses de los empleados que no estén en el archivo original. Si el empleado ya tuviera algún plus, se sumarán los pluses de ambos archivos.