Lección 19: Conceptos y operaciones fundamentales sobre ficheros

- Motivación
- Características del almacenamiento masivo
- Formato de ficheros
- Tipos de campos y registros
- Operaciones con ficheros
- Iteración sobre ficheros
- Consideraciones finales

Motivación

Después de estudiar las mejores estructuras de datos para realizar una aplicación de biblioteca, nos percatamos de que tarde o temprano los datos que se almacenan en las estructuras de datos en memoria deberán pasar a fichero:

- Para no tener que alimentar continuamente un ordenador con la aplicación
- Porque puede que no toda la información quepa en memoria principal
- La mayoría de las aplicaciones necesitan manejar ficheros

Necesidad de almacenamiento masivo

- En el ciclo de vida de los datos de un sistema informático se necesitan ficheros para:
  - Mantener datos de entrada, y/o
  - Albergar datos de salida o resultados y/o
  - Guardar momentáneamente datos de intercambio, resultados parciales, etc.
- Estructuras de ficheros
  - Cuando la aplicación maneja tanta información que no se puede almacenar en memoria, es necesario emplear estructuras de ficheros (lecciones 20 y 21)

Tipos de dispositivos

Son periféricos que sirven para almacenar/recuperar información de forma permanente

- **Dispositivos magnéticos:** el disco se divide en pistas concéntricas
  - Discos duros, disquetes, cintas, etc.
- **Dispositivos ópticos:** las pistas están dispuestas en espiral
  - CD, DVD, Blue Ray
- **Memorias flash:** estructura en modo matricial
  - CF, MicroSD, MiniSD, MMC, SD, SM/SMC, etc.

Costes de acceso a disco

El tiempo de acceso es la suma de realizar estas tres operaciones físicas (CDs, DVDs, discos duros, etc)

- Tiempo de desplazamiento
  - Mover el brazo y situarlo en el cilindro adecuado; depende del número de cilindros y del tiempo de arranque de la cabeza
- Retraso por rotación
  - Es el tiempo que hay que esperar para que el disco gire y se sitúe debajo de la cabeza
- Tiempo de transferencia
  - Depende el número de bytes que haya que transmitir

Costes de acceso a disco

Los tiempos de acceso varían de unos dispositivos a otros:

- Los discos ópticos son más lentos porque hay que encontrar los sectores dentro de la espiral
- Un CD tiene unos 270.000 o 330.000 sectores
- La memorias flash no poseen partes móviles ni mecánicas
  - Son silenciosas, de bajo consumo
  - Derivan de las memorias EEPROM
  - Son matrices de transistores
  - Poseen parte de acceso directo y secuencial

Costes de acceso a disco

- Dado que el acceso a los datos de un fichero es mucho más lento, el objetivo de cualquier sistema de ficheros es acceder el mínimo número de veces
  - Es impracticable necesitar cientos de accesos
  - Es poco recomendable necesitar decenas
  - Es conveniente necesitar menos de diez accesos
- Un **bloque de fichero** es la mínima cantidad de bytes que se transfiere en una operación de lectura/escritura entre fichero y M.P. (o viceversa)
  - El tamaño del bloque depende del dispositivo de almacenamiento y del S.O.
  - Suele tener 512, 1024, 2048 o 4096 bytes

Tamaño del bloque

- Dado que como mínimo leemos/escribimos los bytes de un bloque, es muy recomendable:
  - Leer/escribir todos los registros que quepan en un bloque
  - Las estructuras de datos en ficheros deben agrupar estos registros en cubetas y adaptar los algoritmos para que las L/E sean mínimas
  - Las cubetas deben tener un tamaño acorde con el tamaño del bloque
- Para conocer el tamaño del bloque (en linux)
  - `sudo tune2fs -l /dev/sda7 | grep “Block size”`

Estructuras de ficheros

Una aplicación debe estructurar la información en campos y registros para facilitar su manejo

- **Campo**: la unidad básica de información que mantiene un dato sencillo
  - Ejemplos: dni, nombre, edad....
- **Registro**: colección de campos relacionados
  - Ejemplo: empleado (dni, nombre, edad...)
- Un fichero es una colección de registros
- Normalmente una operación de L/E consiste en el traspaso de un registro (o de varios)
- En memoria se mantiene sólo un registro (o varios)

Formato de ficheros

Ficheros de texto:

- Son ficheros legibles y editables (con un editor)
- Utilizan una codificación ASCII del tipo: UTF-8, ISO-8859-1, etc.
- Muy utilizados como archivos de E/S por ser manipulables por otras aplicaciones y plataformas
- Necesitan ser convertidos al pasar a M.P.

Ficheros binarios:

- L/E más rápida por no necesitar transformación
- Más compactos y menos transportables porque son totalmente específicos de una aplicación

Tipos de campos

Campos de longitud fija:

- Cada campo tiene un espacio de tamaño fijo
- Desperdicia espacio pero son más fáciles y rápidos de manipular

4570 10/12/1980 Pedro Jiménez Sanz 953221014

Campos de longitud variable

- Necesitan una marca, un indicador de longitud para almacenarse o un identificador de campo:

044570**10**10/12/1980**18**Pedro Jiménez Sanz**09**953221014

4570|10/12/1980|Pedro Jiménez Sanz|953221014

numlic=4570|fechanac=10/12/1980|nombre=Pedro Jiménez Sanz|tlf=953221014

Tipos de registros

Registros de longitud fija:

- compuestos de campos de longitud fija

```
4570 10/12/1980  Pedro Jiménez Sanz    953221014
4572 11/05/1982  Juan Gómez Prieto     953228012
```

Registros de longitud variable

- Necesitan una marca al final de cada registro, un contador del tamaño o un índice externo

```
4570|10/12/1980|Pedro Jiménez Sanz|953221014$4572|11/05/1982|Juan Gómez Prieto|953228012$
```

```
044 4570|10/12/1980|Pedro Jiménez Sanz|953221014 043 4572|11/05/1982|Juan Gómez Prieto|953228012
```

```
4570|10/12/1980|Pedro Jiménez Sanz|953221014|4572|11/05/1982|Juan Gómez Prieto|953228012
```

00 45 ...

Operaciones con ficheros

Definamos una clase que representa un fichero de tipo registros de Libros

```
class Libro {
    char titulo[80], autores[40];
    char editorial[40], ISBN[10];
    int anio; float precioActual;
public:
...
  void escribir(fstream &stSalida);
  void leer(fstream &stEntrada);
};
```

```
#define MARCA_BORRADO '*'

class FicheroLibro {
    fstream f;
    unsigned long tam;
    friend class IteradorLibro;

public:
    FicheroLibro(const string &name, bool create = false);
    virtual ~FicheroLibro() {};
    bool leer(unsigned long pos, Libro &libro);
    unsigned long insertar(Libro &libro); // Devuelve la
posición
    void borrar(unsigned long pos);
    void modificar (unsigned long pos, Libro &libro);
    IteradorLibro inicio() {
        f.seekg(0);
        return IteradorLibro(*this, 0);
    }
    unsigned long tamanio() { return tam; }
};
```

La clase fichero

La clase Libro se implementa de tamaño constante y se serializa

```
void Libro::escribir(fstream &stSal){
    stSal.write(titulo, sizeof(titulo));
    stSal.write(autores, sizeof(autores));
    stSal.write(editorial, sizeof(editorial));
    stSal.write(ISBN, sizeof(ISBN));
    stSal.write((char*)&anio, sizeof(anio));
    stSal.write((char*)&precioActual, sizeof(precioActual));
}

void Libro::leer(fstream &stEnt){
    stEnt.read(titulo, sizeof(titulo));
    stEnt.read(autores, sizeof(autores));
    stEnt.read(editorial, sizeof(editorial));
    stEnt.read(ISBN, sizeof(ISBN));
    stEnt.read((char*)&anio, sizeof(anio));
    stEnt.read((char*)&precioActual, sizeof(precioActual));
}
```

Leer una posición

```
FicheroLibro::FicheroLibro(const string &name, bool crearNuevo) {
    if (!crearNuevo)
        f.open(name.c_str(), fstream::binary | fstream::in | fstream::out);
    else
        f.open(name.c_str(), fstream::binary | fstream:in |
                             fstream::out | fstream::trunc);
    if (!f) throw ErrorApertura();
    f.seekg(0,ios::end);
    tam = f.tellg();
}
```

- La lectura de un registro se realiza conociendo la posición donde se ubica el dato

```
bool FicheroLibro::leer(unsigned long pos, Libro &libro){
    if (pos >= tam) throw ErrorPosicionIncorrecta();
    char c;
    f.seekg(pos);
    f >> c;
    if (c == MARCA_BORRADO) return false;
    f.putback(c);
    libro.leer(f);
    f.clear();
    return true;
}
```

Inserción/modificación

- La inserción al final del fichero es una función rápida que sólo necesita un acceso

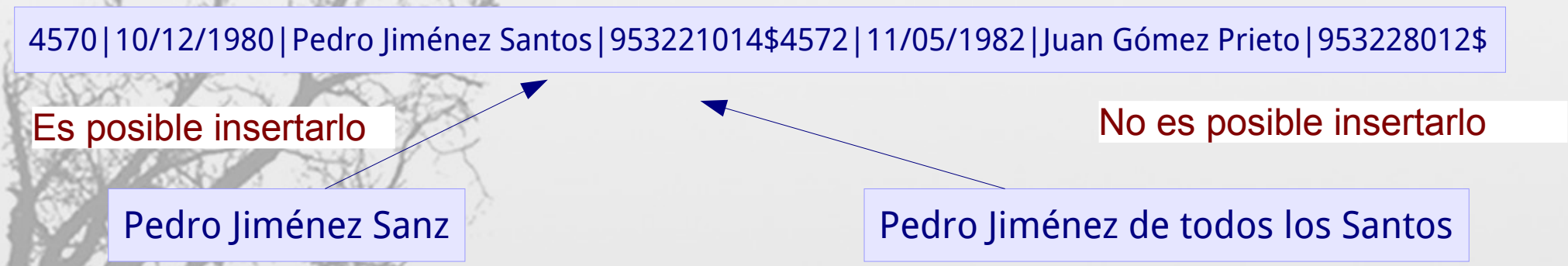
```
unsigned long FicheroLibro::insertar(Libro
&libro){
    f.seekp(0, ios_base::end);
    unsigned long pos = f.tellp();
    libro.escribir(f);
    tam = f.tellp();
    return pos;
}
```

- La actualización de un registro de tamaño fijo es igualmente simple

```
void FicheroLibro::modificar(unsigned long pos, Libro &libro)
{
    if (pos >= tam) throw ErrorPosicionIncorrecta();
    f.seekp(pos);
    libro.escribir(f);
}
```

Inserción/modificación

- La inserción en posiciones intermedias agrandando el fichero no es apta en ficheros por ser ineficiente
- Lo que sí es habitual es reutilizar espacios correspondientes a datos borrados
- La modificación en registros de tamaño variable es compleja porque puede que el dato no quepa
- En caso de que quepa, dejaría espacios inservibles



Borrado

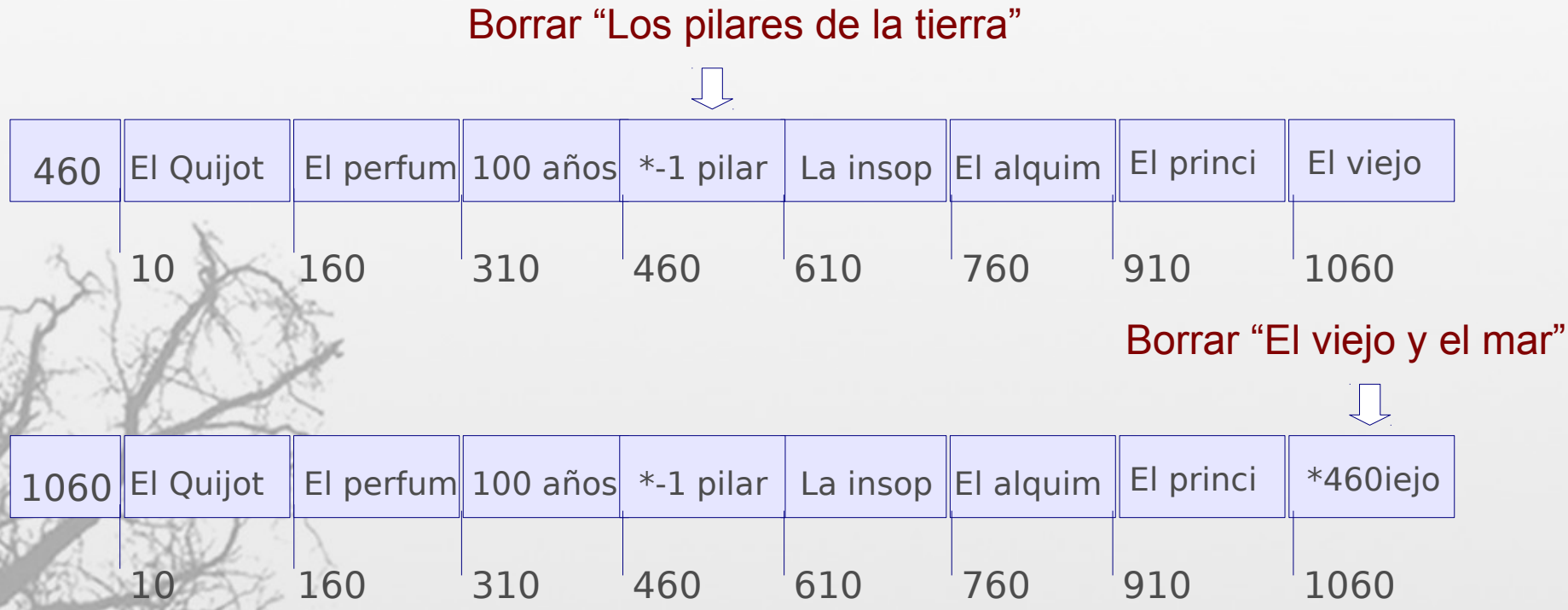
- Los datos no se eliminan físicamente del fichero por ser muy costoso
- Se realiza un borrado lógico que consiste en escribir en dicho registro una marca de borrado

```
void FicheroLibro::borrar(unsigned long pos){
    if (pos >= tam) throw ErrorPosicionIncorrecta();
    f.seekp(pos);
    f << MARCA_BORRADO;
}
```

- Pero lo más recomendable es reutilizar el espacio de los registros borrados, de forma que no se incremente el tamaño del fichero innecesariamente

Borrado

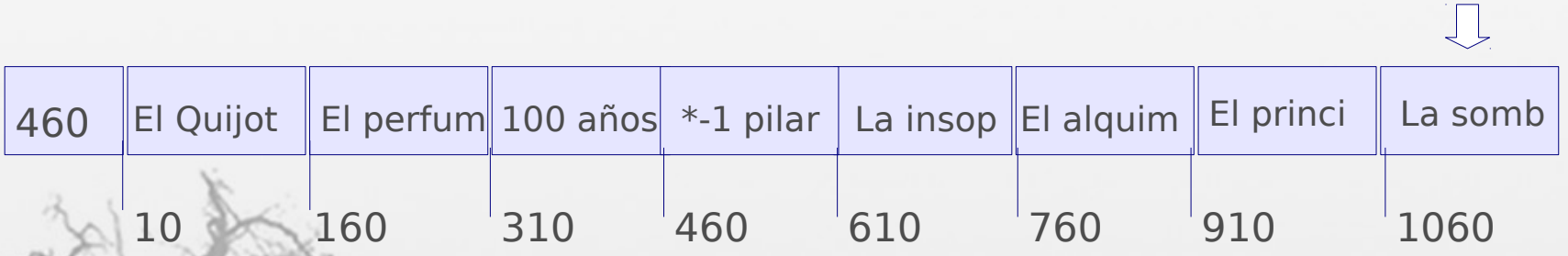
- Para ello se mantiene una pila de borrados en el propio fichero
- La posicion 0 alberga la dirección del primer hueco



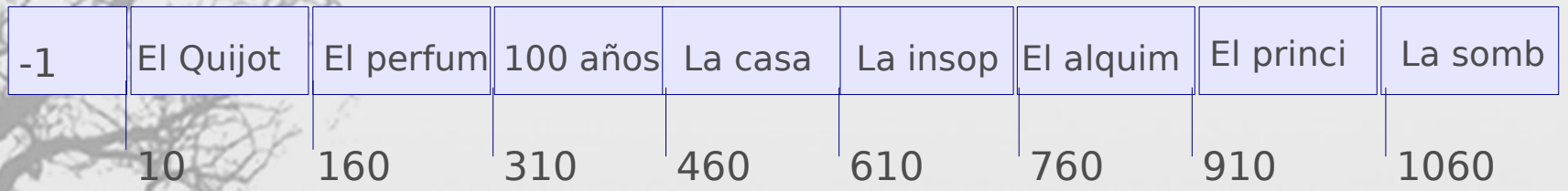
Borrado

- El valor -1 indica el final de la pila
- El nuevo hueco a utilizar fue el último añadido
- Una inserción necesita sólo dos accesos

Insertar "La sombra del viento"



Insertar "La casa de los espíritus"



Búsqueda/ordenación

- Realizar un proceso de búsqueda en un fichero es una tarea inviable (número de accesos > 10)
- Se puede realizar una búsqueda binaria sobre fichero ordenado con $n=10^6 \simeq 2^{20}$ (20 accesos)
- Pero eso implicaría que el fichero no se actualiza porque sería muy costoso (O(n) accesos)
- Por este motivo mantener un fichero ordenado que sufra altas y bajas es impracticable
- Lo que sí se puede realizar es mantener una EEDD en memoria o en otro fichero a modo de índice y que permita acceder al fichero de forma eficiente (Lecciones 20 y 21)

Iteración sobre ficheros

- Aunque la búsqueda sea inviable, a veces es necesario recorrer el fichero (o parte de él)
- Para ello se puede dotar a la clase de iteradores

```
class IteradorLibro {
    FicheroLibro *fLibro;
    Libro l;
    unsigned long p;

public:
    IteradorLibro(FicheroLibro
&fich, unsigned long pos);

    bool haySiguiente();
    void siguiente();

    Libro &libro() { return l; }
    unsigned long pos() {
        return p;
    }
};
```

```
IteradorLibro::IteradorLibro(FicheroLibro &fich,
                unsigned long pos) {
    fLibro = &fich;
    p = pos;
    fLibro->f.seekg(p);
    siguiente();
}

bool IteradorLibro::haySiguiente() {
    return p < fLibro->tam;
}

void IteradorLibro::siguiente() {
    char c;
    do {
        p = fLibro->f.tellg();
        fLibro->f >> c;
        fLibro->f.putback(c);
        l.leer(fLibro->f);
        fLibro->f.clear();
    } while (c == MARCA_BORRADO &&
             haySiguiente());
}
```

Consideraciones finales

- Los ficheros son imprescindibles para mantener la información de forma permanente
- Como el acceso es muy lento, hay que minimizar el número de accesos
- Las búsquedas y ordenaciones sobre los datos de un fichero no se suelen realizar
- Para manejar la información de los ficheros de forma eficiente se utilizan índices como veremos en las siguientes lecciones