Práctica I
OPERACIONES BÁSICAS DE E/S. REGISTROS

- OBJETIVOS
  - ▷ Conocer y utilizar eficientemente las funciones de E/S de C y C++.
  - ▷ Diseñar un formato genérico para almacenar y recuperar registros
  - ▷ Implementar las operaciones de búsqueda, inserción y eliminación de registros.

Índice

1. DESCRIPCIÓN

El objetivo principal de esta práctica es desarrollar unas librerías de funciones para gestionar registros de un archivo con un formato determinado. Estas librerías se utilizarán para crear y mantener diferentes archivos de datos y de índices.

Vamos a crear unos archivo de datos con los que trabajaremos en todas las prácticas de la asignatura. Cada archivo contiene un registro de cabecera que incluye información general, seguido de los registros de datos. Estos **registros son de longitud fija con los campos concatenados**.

El trabajo lo vamos a dividir en dos partes:

1. En la primera parte de la práctica se define un formato para registros y archivos, y se implementan unas librerías de E/S **genéricas** que permiten realizar operaciones de lectura y escritura de registros de cualquier tipo. Para ello se va a usar una zona de memoria que llamaremos buffer, de tamaño adecuado, donde se guarda la información que se va a intercambiar con el disco. También se implementarán las operaciones de mantenimiento: inserción y eliminación de registros (apartados 1 y 2).

2. Usando estas librerías se crean los ficheros de datos binarios con este formato. Sobre estos fichero se realizarán operaciones de búsqueda de registros, eliminación, inserción, visualización, etc. (apartados 3 y 4).

La práctica se realizará en C++. El diseño de clases básico, y los prototipos de los métodos principales se irán proporcionando en las sucesivas sesiones de prácticas a través del webCT.

2. CONOCIMIENTOS PREVIOS NECESARIOS

- **Teoría**: Temas 1 y 2. Construcción de RLF y operaciones (tema 3).
- Memoria dinámica: reserva, operaciones básicas.
- Funciones de manejo de cadenas.
- Clases abstractas. Herencia.
- Clases y métodos de E/S con archivos en C++.
- **Bibliografía**: *File Structures. 3rd Edition.* Folk, Zoellick. Capítulo 4.

3. FORMATOS DE DATOS BÁSICOS

3.1. Buffer de E/S

Para mejorar la eficiencia de las operaciones de E/S, **cada vez que se acceda a un registro en disco se leerá completo**, y no campo a campo. Vamos a utilizar una zona de memoria que llamaremos "buffer de E/S", donde los métodos que procesan los registros pondrán los campos del registro que va a ser escrito en disco (en el formato adecuado). Del mismo modo, cuando se lea un registro de disco se almacenará en esta zona de memoria, y los métodos recuperarán los campos del registro leído.

El Buffer se implementa como una cadena de caracteres. Una vez conocido el formato de los registros de entrada se le reserva el espacio necesario y su **tamaño se mantiene constante** durante toda la ejecución.

3.2. Formato Ficheros Binarios

Los ficheros de salida que se crean contienen: (ver figura 1)

- Un primer registro de cabecera de tamaño fijo, con información sobre los registros de datos que contiene el archivo. Se mantiene una copia en memoria en `Header`.
- A continuación se almacenan los registros de datos **de longitud fija**.

3.3. Formato de los Registros

Los registros serán de **longitud fija**. Al principio mantienen un byte de control que indica si el registro es válido (1) o no (0). A continuación, aparecerán los distintos campos del registro, uno a continuación del anterior, concatenados y separados por un carácter delimitador (|).

Para mantener la lista de registros eliminados: en los registros borrados (no válidos), se escribirá sobre el primer campo la posición del siguiente elemento de la lista de registros borrados.

Por ejemplo, el primero es un registro válido, y el segundo es el mismo registro que ha sido eliminado:

1 |**TEATRO COMPLETO VOL. 3**|...|**ARISTOFANES,**|**1984**|**379**|**Teatro**|**10530041**
0 |**248** |...|**ARISTOFANES,**|**1984**|**379**|**Teatro**|**10530041**

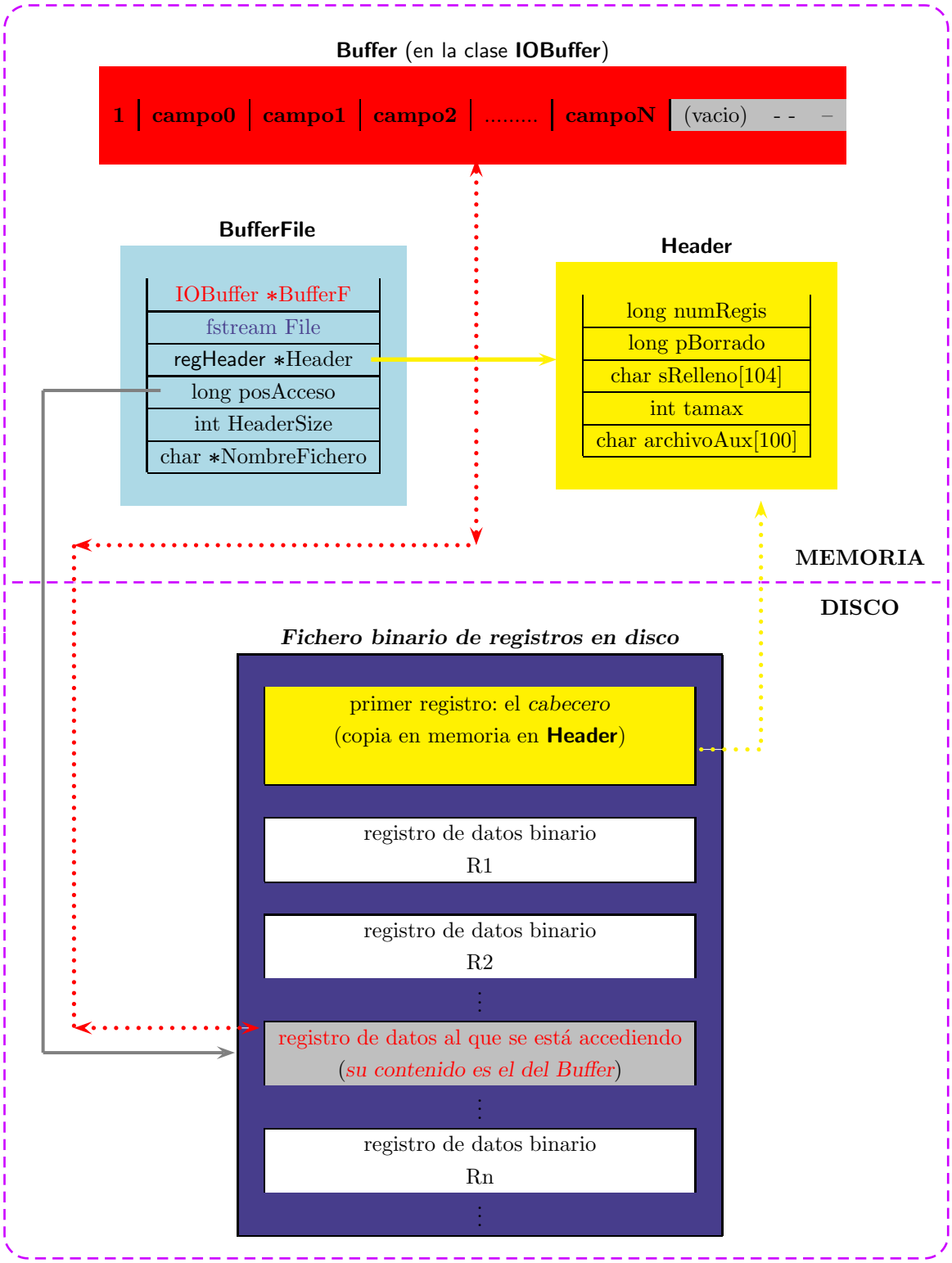


Figura 1: Archivo en disco y buffer de memoria asociado.

3.4. Jerarquía de Clases

Se definen tres clases para operar con el buffer:

- **IOBuffer**: clase abstracta. Contiene los métodos para realizar la lectura/escritura en disco del buffer de E/S. También se definen los métodos que guardan/recuperan los campos de cualquier registro en el buffer.
- **RLFenBuffer**: clase derivada de la anterior que implementa las operaciones de R/W de registros de longitud fija (RLF).
- **CampoDel_Buffer**: clase derivada de la anterior que pone en el buffer los campos de los registros separados con un carácter delimitador.

Y para manipular los archivos y registros de datos:

- **BufferFile**: contiene métodos para crear, leer o escribir cualquier fichero usando un buffer para E/S. Las funciones son **independientes** del tipo de formato del registro que se lea o escriba.
- **REGISTRODatos**: manipula los registros reales de datos. Esta clase conoce la estructura de cada campo y la del registro completo.

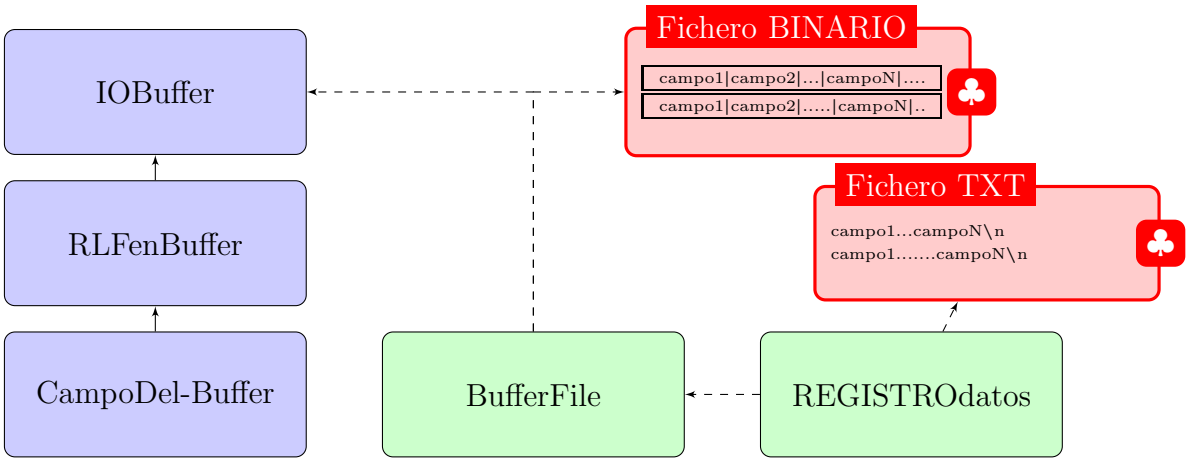


Figura 2: Jerarquía de clases y archivos utilizados.

Las librerías que contienen estas clases (.h) se podrán descargar del ***aula virtual***.

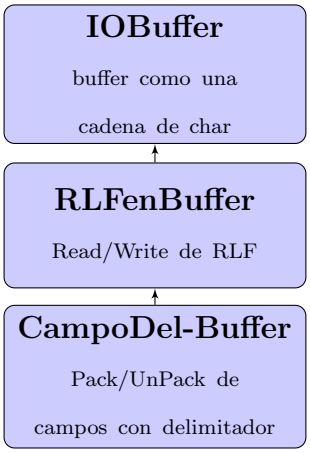
4. APARTADOS A DESARROLLAR

4.1. Funciones para gestionar el buffer de E/S

Recordemos que siempre vamos a asociar a cada archivo de disco una zona en memoria o buffer, de tamaño adecuado a los registros.

Implementar la jerarquía de clases (descritas en el apartado anterior) que gestionan el buffer de E/S como zona de almacenamiento intermedio durante las operaciones de E/S de registros. Las clases básicas están definidas en:

- E_S_Buffer.h : clases para gestionar el buffer. No conocen la estructura del registro:



- Temporización : semana 1 (16 de marzo de 2009).

4.2. Archivo Binario. Operaciones.

Implementar las operaciones básicas sobre un archivo de registros.
Son necesarias las clases de gestión del buffer implementadas en el apartado anterior.
Las nuevas clases que vamos a utilizar son:

- BufferFile.h : clase que gestiona el archivo binario y su buffer de E/S asociado. No conocen la estructura del registro.
- Los archivos tienen un primer registro de cabecero. Se define en esta librería como struct regHeader.
- Se mantiene una lista enlazada de registros eliminados, implementada como una COLA. (Consultar apuntes de clase, tema 3).

Las **operaciones** básicas a implementar son:

1. *Leer/escribir* un registro en el archivo binario ➜ Los registros incorporan un primer byte de control que indica si el registro es válido o no. A continuación, concatenados, los campos de datos:

```
.....
1 |TEATRO COMPLETO VOL. 3|...|ARISTOFANES,|1984|379|Teatro|10530041
1 |TRAGEDIAS ATICAS Y TEBANAS|...|EURIPIDES,.|1991|503|Teatro|10520042
1 |INTRODUCCION A LA LOGICA|...|TARSKI,ALFRED|1985|285|Ciencias|10510043
.....
```

2. *Eliminación* de registros ➜

   Se marcan como "eliminados" usando el byte de control con valor a 0. Seguidamente, se escribirá sobre el primer campo del registro borrado la posición del siguiente elemento de la lista de registros eliminados. (Ver ejemplo sección 3.3)

3. *Inserción y Actualización* de registros ➜

   Si existen registros "eliminados", insertaremos el registro en la primera posición de la lista de registros eliminados (que se mantiene como una pila). En otro caso, insertaremos al final del fichero.
   Se usará el campo pBorrado de la cabecera para mantener la posición del primer elemento en la lista de registros eliminados.

- Temporización :
  - Semana 2: (23 de marzo de 2009)
    1. Métodos generales y operaciones de leer/escribir.
    2. Crear pequeño **programa de prueba** y comprobar funcionamiento.
  - Semana 3: (30 de marzo de 2009). Operaciones de Eliminación e Inserción.

4.3. Funciones para gestionar los registros de datos

Implementar las clases que trabajan con datos de registros "reales".

Estas clases conocen los campos de los registros y los guardan en el buffer de la clase BufferFile para poder procesarlos en operaciones de inserción, búsqueda, etc.

Los datos reales se tomarán de un fichero de texto (en webCT): cada **línea** contiene los campos pertenecientes a un **mismo registro**, separados por un carácter (|). Por ejemplo: "libros.txt" o "coches.txt".

Las clases básicas están definidas en:

- RegistroDatos.h : clases para manipular los registros de datos.
  Conocen el formato de los registros que se guardan en el archivo.

✧ ***Ejemplo:***

Archivo de texto de entrada : "libros.txt"

```
.....
TEATRO COMPLETO VOL. 3|01/08/01|ARISTOFANES,|1984|379|Teatro|10530041
TRAGEDIAS ATICAS Y TEBANAS|01/08/01|EURIPIDES,.|1991|503|Teatro|10520042
INTRODUCCION A LA LOGICA|01/08/01|TARSKI,ALFRED|1985|285|Ciencias|10510043
.....
```

Campos de cada registro de datos:

sTitulo	sFecha	sAutor	iAnyoPublicacion	iNumeroPaginas	sTema	lISBN

Registros de datos en archivo binario de salida:

```
.....
1 |TEATRO COMPLETO VOL. 3|01/08/01|ARISTOFANES,|1984|379|Teatro|10530041
1 |TRAGEDIAS ATICAS Y TEBANAS|01/08/01|EURIPIDES,.|1991|503|Teatro|10520042
1 |INTRODUCCION A LA LOGICA|01/08/01|TARSKI,ALFRED|1985|285|Ciencias|10510043
.....
```

- Temporización : semana 4 (13 abril de 2009).

4.4. Programa principal: creaArchivoBD

Realizar un programa llamado `creaArchivoBD` que lea (línea a línea) los datos del fichero de texto de entrada, los convierta al nuevo formato de registros, y los inserte en el nuevo fichero binario de salida (tras el registro *cabecera*).

Una vez creado, debe realizar las siguientes **operaciones** sobre el archivo:

1. Visualización del archivo de datos (total o parcial)
2. Buscar registros con un determinado valor para uno de sus campos
3. Eliminar un registro dada su posición
4. Añadir nuevos registros

- Temporización : semana 5 (20 abril de 2009).

5. DESARROLLO DE LA PRÁCTICA

- La práctica debe compilar sin errores y funcionar en un entorno de programación **Linux** compatible (CYGWIN, RedHat, Fedora, Debian, etc).

- **Trabajo en Grupo**:

  Las prácticas se realizan en grupos de 4 personas, siguiendo la metodología de trabajo en grupo propuesta para las prácticas de la asignatura. Ésta se puede consultar en un documento publicado en el aula virtual.

- **Trabajo Individual de cada miembro del Grupo**: Cada miembro del grupo debe implementar su propio código, y así será evaluado. Los apartados **1 y 2 se realizarán en grupo**, y los **3 y 4 de forma individual**, de modo que cada miembro del grupo implementará un fichero de datos de salida diferente.

6. RESUMEN DE LA TEMPORIZACIÓN

Cada semana se irán publicando en el webCT las nuevas funciones y/o tipos de datos que habrá que ir añadiendo a las librerías.

semana	Apartado	Tarea	Librerías
1 (16 marzo)	1	buffer de E/S	E_S_Buffer.h
2 (23 marzo)	2	Operaciones de E/S. Programa de prueba	BufferFile.h
3 (30 marzo)	2	Inserción y eliminación	BufferFile.h
4 (13 abril)	3	Registros de datos	RegistroDatos.h
5 (20 abril)	4	Programa principal	

7. DOCUMENTACIÓN A ENTREGAR

Cada grupo debe entregar el siguiente material:

- Código documentado de las clases implementadas (.cpp) (apartados 1 y 2).
- Código fuente documentado y ejecutable de los diferentes programas (`creaArchivoBD`), tipos de datos, archivos de salida, etc. creados por cada miembro del grupo (apartados 3 y 4).

- Una breve memoria que incluya:
  - Diseño y jerarquía de clases utilizadas en cada parte de la práctica.
  - Descripción de los métodos más importantes de esas clases, especialmente los que implementen las operaciones básicas de inserción, eliminación, etc.
  - Material adicional utilizado en el desarrollo de la práctica.
  - Cuadro con la planificación y desarrollo de tareas que ha realizado el grupo.
  - Principales dificultades que se han encontrado: falta de base teórica, problemas de programación, etc.

8. EVALUACIÓN

La máxima puntuación de esta práctica es 15 puntos. Estos son los aspectos que se califican:

1. **Implementación** de las librerías, la eficiencia y calidad del código y su documentación, así como la **originalidad** del mismo.

   Puntuación: **hasta 10 puntos**.

2. **Sesiones de coordinación** con el profesor de prácticas. Se evaluará tanto a nivel **individual** como a nivel de **grupo**.

   Se evalua la coordinación y participación de todos los componentes del grupo , y especialmente el ajuste a la **temporización** y la **calidad del código** implementado.

   Puntuación:
   hasta 2.5 puntos trabajo individual y **hasta 2.5 puntos trabajo grupo**.

9. FECHA DE ENTREGA

24 abril de 2009.