# PRÁCTICA 7: 1 sesión

**(del 23 al 27 de mayo de 2005)**

## GENERACIÓN DE ÍNDICES ANALÍTICOS DE DOCUMENTOS

Vamos a realizar un programa que genere automáticamente el índice analítico de un documento electrónico dado.

Una vez generado el índice tendremos la posibilidad de mostrar todo el índice, buscar una palabra concreta o mostrar las palabras que hemos desechado y no hemos incluido en el índice.

### *0. OBJETIVO*

El objetivo de esta práctica es implementar en C++ el tipo abstracto de datos "Arbol Binario de Búsqueda" y utilizarlo en una aplicación concreta.

### *1. INTRODUCCIÓN*

Un índice analítico de un documento es un índice en el que se incluyen las palabras que aparecen en el texto, el número de veces que aparece la palabra y las páginas del texto en las que aparece. Un ejemplo de un índice analítico podría ser el siguiente:

```
…
MEMORIA (1) -> 2
ORDENADOR (3) -> 1, 3
RAM (5) -> 1, 2, 8
…
```

La palabra 'MEMORIA' aparece una sola vez en el texto, y lo hace en la página 2. La palabra 'ORDENADOR' aparece tres veces, y lo hace en las páginas 1 y 3. Así sucesivamente.

La idea es generar este índice de forma automática a partir de un fichero de texto.

Si analizamos un poco el problema, surge un pequeño inconveniente: al generarse de forma automática el índice hay muchas palabras que aparecen muy frecuentemente (como "el", "la", "y" "que") y que aparecerán siempre en el índice. También habrá determinadas palabras que se repetirán con excesiva frecuencia en documentos que traten de temas concretos, como, por ejemplo, en un documento sobre árboles las palabras *árbol*, *nodo*, *hijo*, etc. A estas palabras que carecen de significado en sí mismas, o que se repiten con excesiva frecuencia se las denomina habitualmente *palabras vacías* (no tienen significado) y sería conveniente que estas palabras no apareciesen en el índice.

Para evitar la aparición de determinadas palabras, estableceremos un valor umbral de apariciones De manera que sólo incluiremos en el índice aquellas palabras que aparezcan en el documento un número de veces inferior al valor umbral.

La implementación de la práctica se realizará mediante dos árboles binarios de búsqueda, uno con las palabras del índice y otro con las palabras vacías. Esta implementación tiene un rendimiento mucho mejor que si lo hiciésemos con listas.

### *2. REALIZACIÓN DE LA PRÁCTICA*

Realizaremos un programa que constará del siguiente menú de opciones:

1.- Generar índice.

2.- Buscar una palabra en el índice.

3.- Mostrar índice.

4.- Mostrar palabras vacías.

0.- Salir del programa.

Evidentemente sólo podremos acceder a las opciones '2', '3' y '4' tras haber generado el índice.

### 1. Generar índice

La opción 1 pedirá un nombre del fichero del que queremos obtener el índice y generará el índice analítico del documento y el conjunto de palabras vacías.

Para realizar esta tarea nos ayudaremos de dos árboles binarios de búsqueda donde iremos insertando las palabras que se vayan encontrando en el documento. Un primer árbol que llamaremos '`indice`', en el que guardaremos las palabras del índice propiamente dicho, y un segundo árbol que llamaremos '`vacias`', en el que guardaremos las palabras vacías.

Cada nodo del árbol binario de búsqueda almacenará un objeto de la clase `RegistroPalabra`. Dichos objetos contendrán la palabra que aparece en el índice, el número de apariciones de la palabra en el texto y una lista con las páginas en las que ha aparecido la palabra.

El proceso que se seguirá en esta opción será el siguiente:

- Se solicitará por teclado el nombre del fichero que queremos indexar y el umbral (valor mínimo de apariciones) para considerar que una palabra es *palabra vacía*.
- A partir del nombre proporcionado por el usuario, se abrirá el fichero y se leerán las palabras del documento. El fichero se leerá carácter a carácter y se irán conformando las palabras, teniendo en cuenta que una palabra está formada por una secuencia de los siguientes caracteres: '`a`' a '`z`', '`A`' a '`Z`', '`ñ`', '`Ñ`' y las vocales acentuadas. Todas las palabras se deberán transformar en mayúsculas.
- Para cada palabra del documento de entrada realizaremos las siguientes tareas:
  - Se comprueba si es la palabra **`SalPag`**, que es el indicador de cambio de página. Si es así se incrementa el contador de páginas.
  - Sino
    - Se comprueba si la palabra aparece en el árbol de palabras vacías. Si es así, se ignora esta palabra y se lee la siguiente.
    - Si no es así, se busca en el índice analítico para ver si ya contiene la palabra. Si la palabra no está en el índice se añade junto con el número de página en la que ha aparecido y se inicializa el número de apariciones a uno. Si la palabra se encuentra en el índice, simplemente se incrementa el número de apariciones y si es necesario se actualiza la lista de números de página en las cuales aparece la palabra (si una palabra aparece varias veces en la misma página, esa página sólo debe de aparecer una vez en la lista de apariciones de la palabra). Si al incrementar el número de apariciones se rebasa el umbral establecido para las palabras vacías, la palabra se borra del árbol `indice` y se inserta en el árbol de palabras vacías, `vacias`.

### 2.- Buscar una palabra

La opción 2, se limitará a pedir una palabra por teclado y la buscará en el índice. Si la palabra está en el índice mostrará su información (número de apariciones y listado de páginas).

### 3.- Mostrar índice.

La opción 3 mostrará el índice analítico completo, tal y como se ha mostrado previamente en el ejemplo.

### 4.- Mostrar palabras vacías.

La opción 4 mostrará el listado de las palabras vacías del árbol `vacias`.

## *3. DETALLES DE IMPLEMENTACIÓN*

Para la realización de la práctica necesitaremos la utilización de tres clases: La clase `RegistroPalabra`, la clase `ABB` y la clase `Lista`.

### Clase `RegistroPalabra`

La interfaz de la clase `RegistroPalabra` ha de ser la siguiente:

```
class RegistroPalabra
{
    public:
        //Constructor
        RegistroPalabra ();
        //Constructor
        RegistroPalabra (string);
        // Devuelve el número de apariciones de la palabra
        int Apariciones ();
        // Devuelve la palabra
        string Palabra ();
        // Devuelve la última página en la que apareció la palabra
        int UltimaPagina ();
        // Muestra la información del RegistroPalabra
        void MostrarRegistroPalabra ();
        // Actualiza el número de apariciones (+1) y si está en una nueva
        // página (argumento) la introduce
        void ActualizarPaginas (int);
    private:
         // Palabra del índice
         string palabra;
         // Número de veces que ha aparecido la palabra
         int apariciones;
         // Lista con las páginas en las que ha aparecido la palabra
         Lista paginas;
         // Convierte un carácter a mayúscula
         char AMayuscula (char c);
};
```

### Clase `Lista`

En la clase `RegistroPalabra` se emplea la clase `Lista`. Utilizaremos la clase lista proporcionada por el profesor en la práctica anterior. Para poder utilizarla, sólo tendremos que asegurarnos que el tipo Valor de la lista es entero:

```
typedef int Valor;
```

### Clase `ABB`

La clase `ABB`, árbol binario de búsqueda, se implementa tal y como se ha visto en las clases teóricas, haciendo que `ValorABB` sea `RegistroPalabra` y añadiendo un nuevo método `InsertarModificar` que añada la palabra al árbol de índices si la palabra no había aparecido previamente o modifique adecuadamente los valores contenidos en el árbol (incrementar el contador de apariciones y actualizar la lista de páginas) si la palabra ya estaba en el índice. (**Ayuda**: Para insertar en el árbol `indice` emplearemos el método `InsertarModificar` y para insertar en el árbol `vacias` emplearemos el método habitual, `Insertar`).

Con esto la interfaz de la clase quedaría:

```
typedef RegistroPalabra ValorABB;
typedef string Clave;

class ABB
{
    public:
        ABB ();
        ABB (const ABB&);
        ~ABB ();
        const ABB& operator= (const ABB&);

        bool Informacion (ValorABB &);
        ABB& HijoIzdo ();
        ABB& HijoDcho ();
        bool ABBVacio ();

        // Busca una clave (palabra)
        ABB& Buscar (Clave);
        // Insertar una cierta clave, aparecida en una cierta pagina
        bool Insertar (Clave, int);
        // Borra del árbol una clave (palabra)
        bool Eliminar (Clave);

        // Insertar o modificar la información asociada a una cierta clave,
        // aparecida en una cierta pagina
        bool InsertarModificar (Clave, int);

   private:
        typedef ABB* PunteroABB;

        ValorABB info;
        PunteroABB izdo;
        PunteroABB dcho;
        bool esVacio;

        bool Copiar (const ABB&);
        void Vaciar ();
};
```

Cabe destacar que a pesar de que la información que se guarda en el árbol es una estructura compleja, tanto la búsqueda como la inserción y la eliminación se realizan atendiendo a una parte concreta de esta información a la que llamamos '`Clave`' (que en nuestro caso es la palabra que aparece en el índice) y que identifica unívocamente la información de cada nodo.

El entero asociado a `Insertar` y a `InsertarModificar` es la pagina donde se ha encontrado la palabra que queremos insertar en el árbol.

### *4. DETALLES DE LA REALIZACIÓN DEL FORMULARIO*

En el formulario referente a está práctica irán reflejados los siguientes detalles del programa:

- **Las tres clases que se van a utilizar en la práctica:**

  Clase `Lista` / Clase `RegistroPalabra` / Clase `ABB` (aunque la *clase Lista* no la hayamos desarrollado nosotros vamos a utilizar sus métodos públicos y debemos documentarlos).

- **Las funciones en que vamos a dividir el programa principal y que vamos a utilizar a lo largo de nuestro programa, especificando claramente las entradas y salidas de las mismas, así como la tarea que desarrolla explicada muy brevemente.**

En esta práctica se recomienda dividir el programa principal en fundamentalmente cuatro funciones que atiendan a cada una de las tareas del menú.

- **Los diagramas de flujo tanto del programa principal como de aquellas funciones que consideremos más interesantes o complejas.**

  En esta práctica se recomienda la realización del diagrama de flujo de la función principal, teniendo en cuenta que tenemos que atender un menú, y que ciertas opciones del menú sólo pueden realizarse si se ha pasado por la opción 1, así como de la función que atiende la primera opción del menú..

### *5. ENTREGA DE PROGRAMAS*

Cinco días después de realizada la sesión de prácticas se entregará al profesor la siguiente información:

1. Archivo con el programa (**`pr07##.cpp`**)
2. Archivo con la interfaz de la clase `RegistroPalabra` (**`RegistroPalabra##.h`**)
3. Archivo con la implementación de la clase `RegistroPalabra` (**`RegistroPalabra##.cpp`**)
4. Archivo con la interfaz de la clase `ArbolBB` (**`ABB##.h`**)
5. Archivo con la implementación de la clase `ArbolBB` (**`ABB##.cpp`**)

dónde **##** es el número asignado a la pareja (01, 02,...) en cada grupo de prácticas.

Al comenzar la siguiente sesión de prácticas se entregará al profesor el formulario con la documentación definitiva de la práctica reflejando todo el trabajo realizado realmente en la práctica.

## Nota Muy Importante

**Antes de poder empezar a realizar cualquiera de las prácticas es necesario presentar las hojas de especificación de programas (formulario de documentación de programas) con las tareas que se van a realizar en la práctica, explicando brevemente como se van a solucionarse los problemas que se plantean.**

**ENTREGA DE PROGRAMAS:**

**Los días 28, 29, 30, 31 de mayo y 1 de junio de 2005, correspondientes con las fechas de realización de prácticas 23, 24, 25, 26 y 27 de mayo respectivamente.**

**ENTREGA DE FORMULARIOS DEFINITIVOS DE DOCUMENTACIÓN:**

**Los días 30, 31 de mayo, 1, 2 y 3 de junio de 2005, correspondientes con las fechas de realización de prácticas 23, 24, 25, 26 y 27 de mayo respectivamente.**