# MEMORÍA PROYECTO C++

ALBERTO GÁLVEZ GÁLVEZ, DNI: 49448576W
(TITULAR CUENTA, USUARIO: B126)

JAIME SÁNCHEZ SÁNCHEZ, DNI: 48754213D

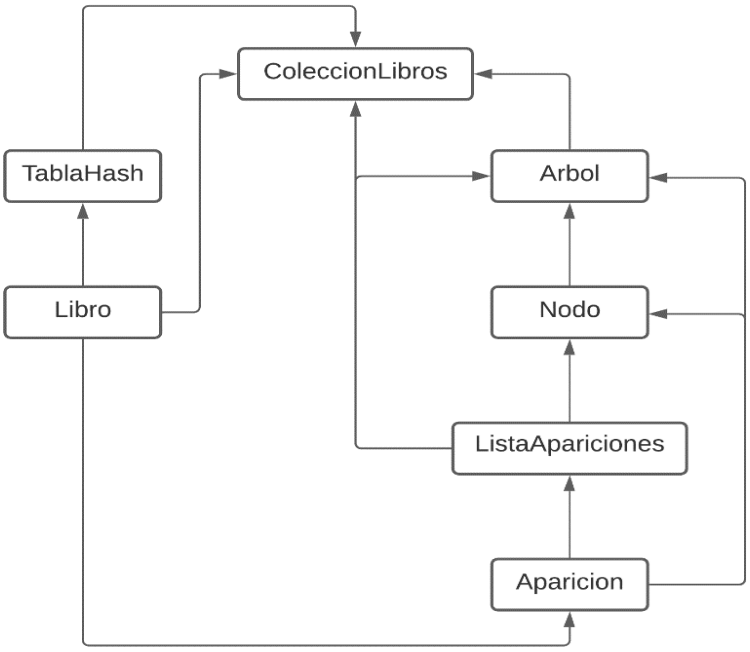
2º INGENIERÍA EN INFORMÁTICA
SUBGRUPO 2.3

12-12-2020

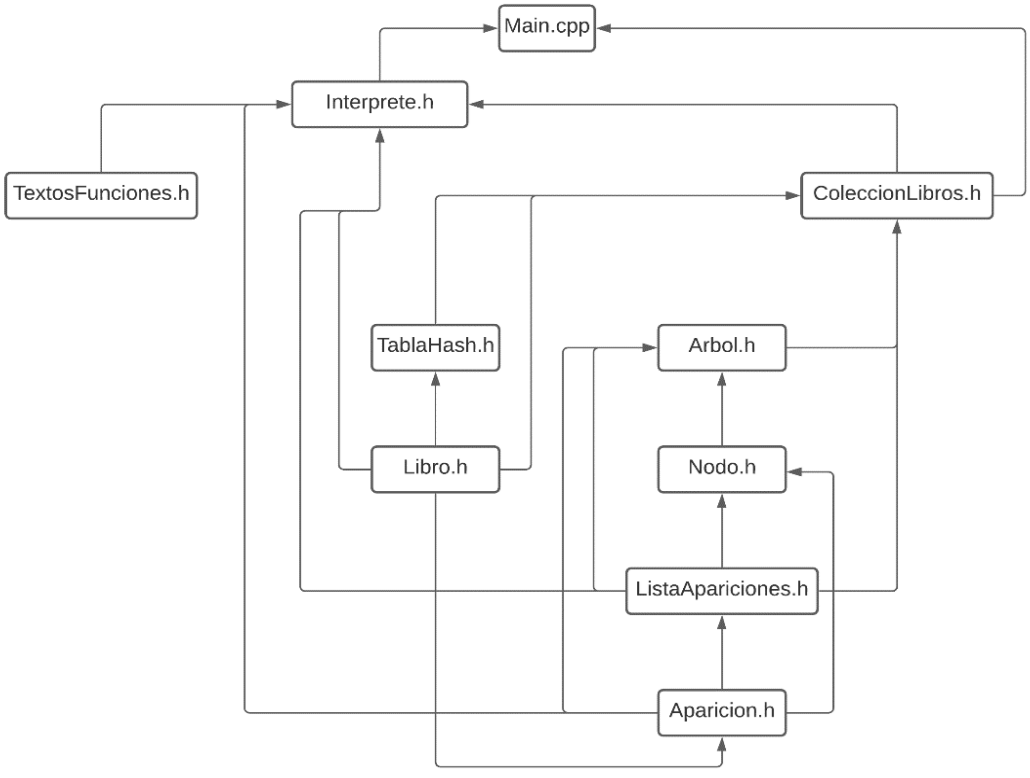
# ÍNDICE

# Análisis y diseño del programa

## Relación entre clases




## Relación entre módulos



# Tablas de dispersión

Optamos por usar en la versión final de nuestro programa, dispersión cerrada. Esta decisión fue tomada en base a un proceso de pruebas (realizadas con el fichero de entrada del ejercicio 201), en el cual, para cada tipo de tabla de dispersión, evaluamos mediante la media para numerosas ejecuciones del programa, como varía su comportamiento de acuerdo a cambios en sus características.

## Dispersión abierta

En primer lugar, testamos como varía la eficiencia en función de si las listas asociadas a cada cubeta son ordenadas o no ordenadas.

| | |
|---|---|
| Listas ordenadas | 0.55 s |
| Listas no ordenadas | 0.53 s |

En segundo lugar, testamos las distintas funciones de dispersión con listas ordenadas (ya que para ciertas operaciones que en un futuro podría decidirse implementar, sería más eficiente la implementación con estas).

| | |
|---|---|
| Método de la división | 0.211 s |
| Método iterativo | 0.189 s |

*La lógica del método iterativo puede verse en el listado de código del anexo.

Como cabría esperar para una aplicación de esta naturaleza (almacenamiento de un número de elementos muy variable), realizamos restructuración. Esta se realiza si el número de elementos insertados duplica al número de cubetas de la tabla. Hemos optado por esta estrategia, para así evitar, por una parte, secuencias de búsqueda de más de 2 unidades de longitud (pese a que no sean demasiado largas, debido a los numerosos accesos a memoria debido a las numerosas operaciones con punteros, son operaciones lentas), así como reestructurar demasiado frecuentemente (ya que se trata de una operación bastante costosa por lo comentado anteriormente).

La restructuración se realiza en los siguientes pasos: 1) Creación de una nueva tabla hash con el doble de cubetas que la anterior. 2) Para cada uno de los elementos de cada una de las cubetas, insertarlo en la nueva tabla y eliminarlo de la anterior. 3) Liberar la tabla antigua, la cual hemos dejado en el paso dos sin elementos, para así evitar que al llamar a su destructor libere los libros que se encuentran ahora en nuestra tabla nueva.

En cuanto a la liberación de la tabla, esta consiste simplemente en llamar al destructor del array de cubetas, que automáticamente llama al destructor de sus elementos (*listas de libros*, módulo implementado por nosotros).

## Dispersión cerrada

Testamos simultáneamente las distintas funciones de dispersión como de redispersión.

| | Doble | Lineal | Salto constante | Cuadrática |
|---|---|---|---|---|
| Método de la división | 0,187 s | 0,205 s | 0,202 s | 0.205 |
| Método iterativo | 0,175 s | 0,181 s | 0,187 s | 0.193 |

*La lógica del método iterativo puede verse en el listado de código del anexo.

Como cabría esperar para una aplicación de esta naturaleza (almacenamiento de un número de elementos muy variable), especialmente en dispersión cerrada, realizamos restructuración. Esta se realiza si el número de elementos insertados supera el 75% del número de cubetas de la tabla. Hemos optado por esta estrategia, para así evitar, por una parte, secuencias de búsqueda demasiado largas, y por otra, reestructurar demasiado frecuentemente (ya que se trata de una operación costosa).
La restructuración se realiza en los siguientes pasos: 1) Creación de una nueva tabla hash con el doble de cubetas que la anterior. 2) Para cada uno de los elementos de cada una de las cubetas, insertarlo en la nueva tabla. 3) Hacer todas las cubetas de la tabla apuntar a NULL y entonces liberar la tabla antigua, para así evitar que, al llamar a su destructor, se liberen los libros que se encuentran ahora en nuestra tabla nueva.

En cuanto a la liberación de la tabla, esta consiste simplemente en llamar al destructor del array de cubetas, que automáticamente llama al destructor de sus elementos (libro).

## Implementación final

Cabe destacar que la función de dispersión iterativa, pese a ser O(log(n)) frente al método de la división O(1), ofrece resultados muy superiores en cuanto a rendimiento, especialmente en dispersión cerrada, donde el problema del amontonamiento se penaliza mayormente. Estos resultados se deben a que el método de la división es muy poco entrópico en comparación al iterativo expuesto, produciéndose secuencias de búsqueda mucho más largas. Véase también que, la única estrategia de redispersión expuesta que evita el amontonamiento, es la redispersión doble, siempre que los sinónimos (con igual valor h(k)) produzcan distinto valor de C(k). Obsérvese que nuestro número de cubetas inicial (16384) elegido, viene motivado por el conocimiento a priori de que, el número de libros a almacenar va a ser aproximadamente de 9000 en el ejercicio 201 y de 1.000.000 en el ejercicio 305. Así mismo, nótese que el número de cubetas es siempre cooprimo con h(k), ya que el número de cubetas es siempre potencia de 2, y h(k) es siempre un número impar. Por todo esto, nos parece adecuado el número de cubetas elegidas para este contexto.

Como conclusión, como se puede ver en las pruebas expuestas anteriormente, podemos concluir que la mejor elección posible contemplada en nuestras pruebas es: una tabla hash cerrada, con función de dispersión iterativa, con función de redispersión doble, con 16384 cubetas en el momento de su creación, y con restructuración.

# Árboles

## Reparto de funcionalidad entre módulos

Hacemos uso de una clase *Arbol*, la cual hace uso de otra clase *Nodo*, desconociendo la primera la implementación de la segunda. La clase *Arbol*, lleva a cabo las distintas inserciones de las palabras y sus apariciones, así como la búsqueda de las apariciones asociadas a una palabra. Para ello, se vale de la clase *Nodo*, que implementa las operaciones para insertar un carácter, consultar un carácter, consultar si un nodo es fin de palabra, e insertar una aparición en la lista de apariciones de un nodo fin de palabra.

## Implementación final

Tan solo hemos probado la implementación con árboles trie, ya que estos cuentan con diversas ventajas que los hacen una mejor alternativa frente a los árboles AVL, como son:

- La búsqueda de la clave es O(m), siendo m la longitud de la palabra a buscar, mientras que en un AVL $O(m*\log_2(n))$, siendo n el número de palabras insertadas en el árbol.
- Ahorro de memoria, puesto que se requiere menos espacio para las claves, ya que estas no se almacenan explícitamente.
- Evitar numerosas aplicaciones de rebalanceo, aproximadamente una cada nos inserciones.

Una vez elegido el árbol trie, realizamos tanto la implementación de estos tanto con listas como con arrays.

Cabe destacar que, si bien es verdad que para un número pequeño de palabras insertadas (por regla general), el uso de listas es muchísimo más eficiente en cuanto a memoria (que no en cuanto a velocidad) que el uso de arrays, debido a que se evita la fragmentación interna; también es verdad que, a partir de cierto punto (el cual lógicamente se supera con creces en el ejercicio 305), es extremadamente más eficiente la implementación con arrays, ya no solo por su mayor velocidad (ya que, para buscar un carácter en un nodo debemos recorrer la lista asociada hasta encontrarlo, mientras que el acceso con arrays es directo), sino también por el ahorro memoria que se produce (ya que, se produciría poca fragmentación externa, mientras que con listas se malgastaría muchísima memoria debido al puntero extra que se tiene para enlazar los nodos de la lista.

En cuanto a la liberación del árbol, se realiza tan solamente con llamar al destructor del nodo raíz, que debido a la definición recursiva de la clase *Nodo*, se llama a los destructores de sus nodos hijos, este proceso se realiza para todos los nodos recursivamente hasta que se detiene al llegar a un nodo nulo.

Variables globales

Tan solo se ha hecho uso de una variable global *ColeccionLibros*, que almacena tanto la información de los libros, como las palabras que se encuentran en ellos y las apariciones de estas últimas en ellos.

# Informe de desarollo

Comenzamos el proyecto el día 24 de octubre y lo terminamos definitivamente el día 8 de octubre. Durante este mes y medio hemos estado trabajando el proyecto una media de 5 días a la semana. La distribución del trabajo ha sido de la siguiente forma:

| Etapa | Análisis | Diseño | Implementación | Validación | Total |
|---|---|---|---|---|---|
| 1ª | 2 h | 3 h | 5 h | 7 h | 17 h |
| 2ª | 3 h | 4 h | 6 h | 8 h | 21 h |
| 3ª | 2 h | 3 h | 4 h | 4 h | 13 h |

*1ª etapa, desde el ejercicio 001 hasta el 007
*2ª etapa, desde el ejercicio 008 hasta el 201
*3ª etapa, desde el ejercicio 301 hasta el 305

Como podemos ver, la primera y segunda etapa son las que más carga de trabajo tienen. Esto es debido a que en los primeros ejercicios tuvimos diversas complicaciones con el tratamiento de la entrada, pues ninguno de los dos habíamos usado las funciones y librerías necesarias apenas. Asimismo, ninguno de los sabíamos usar la componente orientada a objetos de C++, por lo que hubo una etapa de aprendizaje también. Por su parte, gran parte de las 21 horas de la segunda etapa, vienen por la tabla Hash, ya que tuvimos algunos problemas con los índices del array (operar con punteros nulos, liberar doblemente, problemas con la localidad de variables, etc), que hacían que el programa abortase la ejecución. Pero el principal trabajo en la tabla Hash fue el hecho de tener que realizar tantas implementaciones y realizar tantos tests para poder decantarnos por la mejor.

En la última etapa, teníamos bastante claro cómo implementar cada parte, pero surgieron diversas complicaciones debidas a despistes o errores, más que a la lógica del programa como tal.

En cuanto a la metodología, nos conectábamos ambos virtualmente, compartiendo la pantalla. Uno de los dos escribía el código, mientras que el otro se encargaba de revisar sobre la marcha posibles errores de sintaxis. Cabe destacar que el que se encargaba de hacer el rol de revisor solía encontrar los errores de sintaxis muchísimo más rápido que el que escribe el código. Ninguno tuvimos asignado un rol específico, pero en la tercera y primera etapa fue Alberto Gálvez el encargado de escribir, mientras que Jaime Sánchez lo hizo en la segunda.
La carga de trabajo fue repartida de manera equilibrada entre ambos miembros, aunque el encargado de la comunicación con el profesor de prácticas fue, en todo momento, Alberto Gálvez.

# Conclusiones

Podemos decir que la realización de este proyecto ha sido de gran satisfacción, debido a todo lo que hemos aprendido sobre buenas prácticas, gestión de un proyecto grande (en comparación a lo hecho anteriormente) y a largo plazo, el lenguaje C++, el uso de ficheros de entrada y de salida, tratamiento de errores, y sobre todo sobre el uso de algoritmos y estructuras de datos.

Tenemos la sensación de ahora tener mejores competencias a la hora de tomar decisiones sobre el diseño de una aplicación, así siendo conscientes sobre el tipo de estructuras de datos y algoritmos a elegir dependiendo del problema a tratar. Además, la temática nos del proyecto nos parece todo un acierto, ya que hemos puesto en práctica todos y cada uno de los contenidos que hemos visto en clase de teoría, y además nos parece una temática de algo similar a lo que podríamos encontrar en un contexto laboral.

Por otra parte, el uso del lenguaje *C++* nos ha parecido un poco engorroso, debido a que la gestión de la entrada/salida se nos ha hecho un poco extraña. También, algo que nos ha relentizado mucho ha sido la falta de formación sobre herramientas *debugger*, lo cual habría sido de muchísima ayuda para poder solucionar errores en tiempo de ejecución más fácilmente, los cuales podríamos decir que han formado un tercio del tiempo dedicado a este proyecto.

Cabe decir, que pese a ser un proyecto con una implicación bastante notoria, es preferible a un examen. Esto, no porque un examen sea más difícil o requiera más tiempo de estudio, sino porque la realización de un proyecto como este nos prepara mejor para lo que nos vamos a encontrar en el mundo laboral como trabajadores.

En resumen, nos ha resultado un proyecto de gran interés y utilidad, pero que a la misma vez algunos detalles nos han hecho dedicar demasiado tiempo a errores o cuestiones menores, más que a la lógica del proyecto en sí.

# Anexo

## Envíos plataforma Mooshak

| Ejercicio | Número de envío |
|---|---|
| 001 | 496 |
| 002 | 495 |
| 003 | 569 |
| 005 | 1027 |
| 006 | 1482 |
| 007 | 1759 |
| 008 | 2025 |
| 201 | 3906 |
| 301 | 2831 |
| 302 | 2936 |
| 303 | 3162 |
| 304 | 3502 |
| 305 | 3505 |

*Cabe destacar la evolución que han ido surgiendo los distintos módulos a lo largo del proyecto, llegando incluso algunos a ser eliminados, por ejemplo ListaDeLibros (ya que formaba parte de dispersión abierta, y nuestra implementación final fue con dispersión cerrada).

# Listado de código

Libro.cpp:

```
#include "Libro.h"
#include <iostream>
using namespace std;


Libro::Libro(){
	this->isbn = -1;
	this->ano = -1;
	this->num_capitulos = -1;
	this->titulo = "Sin asignar";
	this->autores = "Sin asignar";
}

int Libro::get_num_capitulos(){
	return this->num_capitulos;
}

//Leer se encarga de establecer los datos de un libro según lo introducido por entrada estándar
void Libro::leer(){
	cin >> this->isbn >> this->ano >> this->num_capitulos;
	string ignorar;
	getline(cin, ignorar);
	getline(cin, this->titulo);
	getline(cin, this->autores);
}

void Libro::mostrarCorto(){
	cout << this->titulo << ", " << this->autores << ", " << this->ano;
}
void Libro::mostrarLargo(){
```

```
	cout << this->titulo << endl << this->autores <<  ", " << this->ano << endl << this->num_capitulos <<  " capítulos";
}

unsigned long long int Libro::get_isbn(){
	return this->isbn;
}
```

Libro.h:

```
#ifndef __LIBRO_H__
#define __LIBRO_H__
#include <iostream>
using namespace std;

class Libro{
	private:
		unsigned long long int isbn;
		int ano, num_capitulos;
		string titulo, autores;

	public:
		Libro();
		int get_num_capitulos();
		void leer();
		void mostrarCorto();
		void mostrarLargo();
		unsigned long long int get_isbn();
};
```

Aparcion.cpp:

```
#include "Aparicion.h"
#include "Libro.h"

// Constructor de la clase donde el puntero a libro el número de capítulos y el número de parrafos de la aparición son pasados como parámetros
Aparicion::Aparicion(Libro * pl, int num_capitulo, int num_parrafo){
	this->pl = pl;
	this->num_capitulo = num_capitulo;
	this->num_parrafo = num_parrafo;
}

// Método encargado de mostrar los atributos de la aparición
void Aparicion::mostrar(){
	this->pl->mostrarCorto();
	cout << ", Cap. " << this->num_capitulo << ", par. " << this->num_parrafo << endl;
}

// Método encargado de definir la operación "menor que" de las apariciones
bool Aparicion::operator<(const Aparicion &otra){
	if( this->pl->get_isbn() < otra.pl->get_isbn() )	return true; // primero se compara por ISBN
	else if( this->pl->get_isbn() > otra.pl->get_isbn() ) return false;
	else{
		if(this->num_capitulo < otra.num_capitulo) return true; // después por capítulos
		else if(this->num_capitulo > otra.num_capitulo) return false;
		else{
			if(this->num_parrafo < otra.num_parrafo) return true; // por último por párafos
			else return false;
		}
	}
}
// Método encargado de definir la operación "igual que" de las apariciones
```

```
bool Aparicion::operator==(const Aparicion &otra){
	if(this->pl->get_isbn() == otra.pl->get_isbn()		// tanto número de capitulos, número de párrafos e ISBN deben ser iguale
			&& this->num_capitulo == otra.num_capitulo
			&& this->num_parrafo == otra.num_parrafo)
		return true;
	else
		return false;
}
```

Aparcion.h:

```
#ifndef __APARICION_H__
#define __APARICION_H__
#include "Libro.h"

class Aparicion{
	private:
		Libro * pl;
		int num_capitulo;
		int num_parrafo;

	public:
		Aparicion(Libro * pl, int num_capitulo, int num_parrafo);
		void mostrar();
		bool operator< (const Aparicion &otra);
	bool operator== (const Aparicion &otra);
};
#endif
```

ListaApariciones.cpp:

```
#include "Aparicion.h"
```

```
#include "ListaApariciones.h"
#include <list>

ListaApariciones::ListaApariciones(){
	this->num_elementos = 0;
}

//Insercion de un nuevo puntero en un la lista
//Los Apariciones estan ordenados por ISBN en la lista
void ListaApariciones::insertar(Aparicion nueva){
	list<Aparicion>::iterator it = this->listaApariciones.begin();
	while( it != this->listaApariciones.end() && *it < nueva )	++it;
	if( it == this->listaApariciones.end() || !(nueva == *it ) ){
		this->listaApariciones.insert(it, nueva);
		this->num_elementos+=1;
	}
}

// Comprobamos si la lista es vacía
bool ListaApariciones::esVacia(){
	return num_elementos == 0;
}

// Método encargado de mostrar las apariciones de una lista de apariciones
void ListaApariciones::mostrar(){
	cout << "Total: " << num_elementos << " resultados" << endl;
	list<Aparicion>::iterator it = this->listaApariciones.begin();
	for(int i = 1; i <= num_elementos; ++i){
		cout << i << ". ";
		(*it).mostrar();
		++it;
	}
}
```

```
// Copiamos en la lista pasada como parámetro
// el contenido de la lista que realiza la instancia
void ListaApariciones::copiarLista(ListaApariciones * copia){
        list<Aparicion>::iterator it = this->listaApariciones.begin(); // iterador para recorrer la lista a copiar
    for(int i = 0; i < num_elementos; ++i){
        copia->listaApariciones.push_back(*it);
            ++it;
    }
    copia->num_elementos = this->num_elementos;
}

// Se borran de la lista que realiza la instancia, las apariciones
// que no serían fruto de la intersección entre ella y la pasada como parámetro
void ListaApariciones::interseca(list<Aparicion> & lista){    // método privado auxiliar
        list<Aparicion>::iterator it1 = this->listaApariciones.begin();          //iterador de la lista que realiza la instancia
        list<Aparicion>::iterator it2 = lista.begin(); //iterador de la lista pasada como parámetro

        while( it1 != this->listaApariciones.end() ){
                if(it2 == lista.end() || *it1 < *it2){
                        this->listaApariciones.erase(it1++);
                        --this->num_elementos;
                }
                else if(*it1 == *it2){
                        ++it1;
                        ++it2;
                }
                else ++it2;
        }
}
```

```
// Se borran de la lista que realiza la instancia, las apariciones
// que no serían fruto de la intersección entre ella y la pasada como parámetro
void ListaApariciones::une(list<Aparicion> & lista){ // método privado auxiliar
	list<Aparicion>::iterator it1 = this->listaApariciones.begin();		//iterador de la lista que realiza la instancia
	list<Aparicion>::iterator it2 = lista.begin(); //iterador de la lista pasada como parámetro

	while( it2 != lista.end() ){
		if( it1 == this->listaApariciones.end() ){
			this->listaApariciones.push_back(*it2++);
			++this->num_elementos;
		}
		else if(*it1 < *it2)
			++it1;
		else if(*it2 < *it1){
			this->listaApariciones.insert(it1, *it2);
			++this->num_elementos;
			++it2;
		}
		else{
			++it1;
			++it2;
		}

	}
}

// Llama a la función auxiliar que realiza la interseción
void ListaApariciones::interseca(ListaApariciones * lista){
	this->interseca(lista->listaApariciones);
}

// Llama a la función auxiliar que realiza la interseción
```

```
void ListaApariciones::une(ListaApariciones * lista){
        this->une(lista->listaApariciones);
}
```

ListaApariciones.h:

```
#ifndef __LISTA_APARICIONES_H__
#define __LISTA_APARICIONES_H__
#include "Aparicion.h"
#include <list>

class ListaApariciones{
        private:
                list<Aparicion> listaApariciones;       // lista de Apariciones
                int num_elementos;   // numero de elementos insertados
                void interseca(list<Aparicion> & lista);
                void une(list<Aparicion> & lista);
        public:
                ListaApariciones();
                void insertar(Aparicion nuevo);
                bool esVacia();
                void mostrar();
                void copiarLista(ListaApariciones * copia);
                void interseca(ListaApariciones * lista);
                void une(ListaApariciones * lista);
};
```

TablaHash.cpp:

```
include "TablaHash.h"
#include "Libro.h"
#define NUM_CUBETAS_INICIAL 16384
```

```
TablaHash::TablaHash(){
        this->num_cubetas = NUM_CUBETAS_INICIAL;
        this->tablaHash = new Libro*[num_cubetas];
        this->num_elementos = 0;
}

TablaHash::~TablaHash(){
        delete[] this->tablaHash;
}

// Consulta si un libro se encuentra en una tablaHash
// la clave usada es el ISBN
Libro * TablaHash::consultar(unsigned long long int isbn){
        unsigned int indice = hashCode(isbn);
        if(tablaHash[indice] == NULL)
                return NULL;
        if(tablaHash[indice]->get_isbn() == isbn)
                return tablaHash[indice];
        else
                return secuenciaDeBusqueda(isbn, indice);
}

// Inserta el libro en una tablaHash.
// la clave es el ISBN
void TablaHash::insertar(Libro * nuevo){
        if(num_elementos > num_cubetas*0.75)
                reestructurar();

        unsigned int indice = hashCode(nuevo->get_isbn());
        if(tablaHash[indice] == NULL){
                tablaHash[indice] = nuevo;
                ++num_elementos;
        }
```

```
        else{
                secuenciaDeInsercion(nuevo->get_isbn(), indice, nuevo);
                ++num_elementos;
        }
}

unsigned int TablaHash::hashCode(unsigned long long int clave){          // método privado
        unsigned int result = 0;
        unsigned int factor = 17;
        while (clave > 0) {
                result *= factor;
                result += clave % 10;
                clave /= 10;
        }
        result %= num_cubetas;
        return result;
}

unsigned int TablaHash::hashRedispersion(unsigned long long int clave, unsigned int inicio, unsigned int i){ // método privado
        unsigned int resultado = (inicio+i*((2*clave+1)%num_cubetas))%num_cubetas;
        return resultado;
}

Libro * TablaHash::secuenciaDeBusqueda(unsigned long long int isbn, unsigned int inicio){ // método privado
        unsigned int i = 1;      // tamaño de los saltos
        unsigned int indice = hashRedispersion(isbn, inicio, i);      // indice cambiante
        Libro * elem = tablaHash[indice];
        while(elem != NULL){
                if(elem->get_isbn() == isbn)
                        return tablaHash[indice];
```

```
            indice = hashRedispersion(isbn, indice, i);
            ++i;
            elem = tablaHash[indice];
        }
        return NULL;
}

void TablaHash::secuenciaDeInsercion(unsigned long long int isbn, unsigned int inicio,
Libro * libro){  // método privado
        unsigned int i = 1;             // tamaño de los saltos
        unsigned int indice = hashRedispersion(isbn, inicio, i);      // indice cambiante
        Libro * elem = tablaHash[indice];
        while(elem != NULL){
            indice = hashRedispersion(isbn, indice, i);
            ++i;
            elem = tablaHash[indice];
        }
        tablaHash[indice] = libro;
}

void TablaHash::reestructurar(){      // método privado
        int n_num_cubetas_original = this->num_cubetas;
        this->num_cubetas = 2*n_num_cubetas_original;

        Libro ** original = this->tablaHash;
        this->tablaHash = new Libro*[this->num_cubetas];
        Libro * elem;
        unsigned int indice_insertar;
        for(int i = 0; i < n_num_cubetas_original; ++i){
            elem = original[i];
            if(elem != NULL)
                insertar(elem);
        }
```

```
        for (int i = 0; i < n_num_cubetas_original; ++i)        // hacemos NULL todos los elementos de nuestra tabla original
                original[i] = NULL;
        // para así evitar que liberemos los libros
        delete [] original;        // liberamos nuestra Tabla Hash antigua
}
```

TablaHash.h:

```
#ifndef __TABLA_HASH_H__
#define __TABLA_HASH_H__
#include "Libro.h"

class TablaHash{
        private:
                int num_cubetas;       // numero de cubetas
                Libro ** tablaHash;    // array de punteros a libro
                int num_elementos;   // numero de elementos insertados
                unsigned int hashCode(unsigned long long int clave);       // función de dispersión
                unsigned int hashRedispersion(unsigned long long int clave, unsigned int inicio, unsigned int i);       // función de redispersión
                void secuenciaDeInsercion(unsigned long long int isbn, unsigned int inicio, Libro * libro);
                Libro * secuenciaDeBusqueda(unsigned long long int isbn, unsigned int inicio);
                void reestructurar();   // restructura la tabla si num_elementos > 2*num_cubetas
        public:
                TablaHash();
                ~TablaHash();
                void insertar(Libro * nuevo);
                Libro * consultar(unsigned long long int isbn);
};
```

```
#endif
```

Nodo.cpp:

```
#include "Nodo.h"
#include "ListaApariciones.h"
#include "Aparicion.h"
#define TAMANO_ARRAY 27


Nodo::Nodo(){
	this->hijos = new Nodo *[TAMANO_ARRAY];		// array de punteros a nodo
	for(int i = 0; i < TAMANO_ARRAY; ++i)
		this->hijos[i] = NULL; // inicializamos a null los punteros
	this->apariciones = NULL;	// no hay lista de apariciones inicialmente
}

Nodo::~Nodo(){
	delete[] this->hijos;
	delete this->apariciones;
}

Nodo * Nodo::consultaCaracter(char letra){
	if( letra == char(0xC3) )
		return hijos[TAMANO_ARRAY-1];	// la ñ es un caso especial, la
almacenamos en la ultima posicion del array
	else
		return hijos[letra - 'a'];

}

void Nodo::insertaCaracter(char letra){
```

```
        if( letra == char(0xC3) )
                hijos[TAMANO_ARRAY-1] = new Nodo();    // la ñ es un caso especial, la almacenamos en la ultima posicion del array
        else
                hijos[letra - 'a'] = new Nodo();

}

void Nodo::insertaAparicion(Aparicion aparicion){
        if( hayMarca() ) apariciones->insertar(aparicion);    // si ya había una lista de apariciones la insertamos en ella
        else{
                ponMarca();    // si no había una lista la creamos e insertamos la aparición
                apariciones->insertar(aparicion);
        }
}

ListaApariciones * Nodo::obtenerApariciones(){
        if( hayMarca() ) return apariciones;   // comprobamos si hay una lista de apariciones, si la hay entonces la devolvermos
        else return NULL;
}

bool Nodo::hayMarca(){
        return apariciones != NULL;
}

void Nodo::ponMarca(){
        if( !hayMarca() ) apariciones = new ListaApariciones();      // si no había una lista de apariciones, la creamos
}

void Nodo::quitaMarca(){
        delete apariciones;
```

```
        apariciones = NULL;
}
```

Nodo.h:

```
#ifndef __NODO_H__
#define __NODO_H__
#include "ListaApariciones.h"
#include "Aparicion.h"

class Nodo {
   private:
      Nodo ** hijos;  // array de punteros a nodo
      ListaApariciones * apariciones; // lista de apariciones que nos sirve también como marca de fin de palabra
   public:
      Nodo();
      ~Nodo();
      Nodo * consultaCaracter(char letra);
      void insertaCaracter(char letra);
      void insertaAparicion(Aparicion aparicion);
      ListaApariciones * obtenerApariciones();
      bool hayMarca();
      void ponMarca();
      void quitaMarca();
};

#endif
```

Arbol.cpp:

```
#include "Arbol.h"
#include "Nodo.h"
#include "ListaApariciones.h"
#include "Aparicion.h"
#include <iostream>
using namespace std;


// Constructor de la clase Arbol que se encarga de incializar un nodo como raíz
Arbol::Arbol(){
	this->raiz = new Nodo();
}

// Destructor de la clase árbol, llama al destructor del nodo, que a su vez llama recursivamente a todos los nodos de todos los subárboles
Arbol::~Arbol(){
	delete this->raiz;
}

// Método encargado de insertar una aparición de una palabra en el nodo
void Arbol::insertarAparicionPalabra(string cadena, Aparicion aparicion){
	Nodo * actual = raiz;
	int i = 0;
	while( i < cadena.length() ){		// recorremos toda la palabra
		if(actual->consultaCaracter(cadena[i])== NULL)
			actual->insertaCaracter(cadena[i]);			// si no hay una asociación para el caracter actual, la creamos
		actual = actual->consultaCaracter(cadena[i]);
		if( cadena[i] == char(0xC3) ) // si nos encontramos con la ñ avanzamos doblemente, ya que está representada por 2 bytes
			++i;
		++i;
	}
```

```
        actual->insertaAparicion(aparicion); // al final, insertaamos la aparición
}


// Método encargado de obtener una aparición de una palabra en el nodo
ListaApariciones * Arbol::obtenerAparicionesPalabra(string cadena){
        int i = 0;
        Nodo * actual = raiz;
        while( i < cadena.length() && !(actual->consultaCaracter(cadena[i]) == NULL) ){
// avanzamos mientras no sea el final de la cadena o mientras haya una asociación
para el caracter actual
                actual = actual->consultaCaracter(cadena[i]);         // avanzamos en el
nodo
                if( cadena[i] == char(0xC3) ) // si nos encontramos con la ñ avanzamos
doblemente, ya que está representada por 2 bytes
                        ++i;
                ++i;     //avanzamos en la cadena
        }
        if(i < cadena.length() || actual->obtenerApariciones() == NULL)              //
comprobamos si la palabra estaba en el nodo
                return NULL;
        else
                return actual->obtenerApariciones();
}
```

Arbol.h:

```
#ifndef __ARBOL_H__
#define __ARBOL_H__
#include "Nodo.h"
#include "ListaApariciones.h"
#include "Aparicion.h"
#include <iostream>
using namespace std;
```

```
class Arbol {
	private:
		Nodo * raiz;
	public:
		Arbol();
		~Arbol();
		void insertarAparicionPalabra(string cadena, Aparicion aparicion);
		ListaApariciones * obtenerAparicionesPalabra(string cadena);
};


#endif
```

ColeccionLibros.cpp:

```
#include "ColeccionLibros.h"
#include "Arbol.h"
#include "ListaApariciones.h"
#include "Aparicion.h"
#include "Libro.h"
#include <iostream>
#include <sstream>
using namespace std;


ColeccionLibros::ColeccionLibros(){
}

//Inserta un libro en la Coleccion
void ColeccionLibros::insertarLibro(Libro * nuevo){
	tablahash.insertar(nuevo);
}
```

```
// Consulta si un libro se encuentra en la Coleccion, si es así lo devuelve
// si el libro no está devuelve NULL
Libro * ColeccionLibros::consultarLibro(unsigned long long int isbn){
        return this->tablahash.consultar(isbn);
}

void ColeccionLibros::insertarAparicionPalabra(string cadena, Aparicion Aparicion){
    arbol.insertarAparicionPalabra(cadena, Aparicion);
}

ListaApariciones * ColeccionLibros::obtenerAparicionesPalabra(string cadena){
    return arbol.obtenerAparicionesPalabra(cadena);
}

ListaApariciones * ColeccionLibros::interseccionAparicionesDePalabras(string
cadena){
    istringstream palabras(cadena);    // palabras a hacer intersección de sus
apariciones
    string palabra;
    palabras >> palabra;  // primera palabra de la lista de palabras
    ListaApariciones  * aparicionesPalabra = obtenerAparicionesPalabra(palabra);
// la intersección inicial es la lista de apariciones de la primera palabra
    if(aparicionesPalabra == NULL)    // si la lista de apariciones de la primera
palabra es vacía
        return NULL;
    else{   // procedemos a realizar todas las intersecciones
        ListaApariciones * intersecciones = new ListaApariciones();        //
creamos una lista vacía
        aparicionesPalabra->copiarLista(intersecciones);   // copiamos la lista de
apariciones del árbol en nuestra lista de intersección, la cual estaba vacía
        while(palabras >> palabra){  // para cada palabra
            aparicionesPalabra = obtenerAparicionesPalabra(palabra);
            if(aparicionesPalabra != NULL){    // si la lista de la palabra
actual no es vacía
                intersecciones->interseca(aparicionesPalabra);
```

```
                    if( intersecciones->esVacia() ){        // si el conjunto fruto de la intersección es vacío
                        delete intersecciones;
                        return NULL;
                    }
                }
                else{
                    delete intersecciones;
                    return NULL;
                }
            }
            return intersecciones;
        }
}

ListaApariciones * ColeccionLibros::unionAparicionesDePalabras(string cadena){
        istringstream palabras(cadena);       // palabras a hacer union de sus apariciones
        string palabra;
        ListaApariciones * aparicionesPalabra;
        ListaApariciones * uniones = new ListaApariciones();        // creamos una lista vacía
        while(palabras >> palabra){  // para cada palabra
            aparicionesPalabra = obtenerAparicionesPalabra(palabra);
            if(aparicionesPalabra != NULL) {      // si la lista de la palabra actual no es vacía
                uniones->une(aparicionesPalabra);
            }
        }
        if( uniones->esVacia() ){
            delete uniones;
            return NULL;
        }
        else
```

```
        return uniones;
}
```

ColeccionLibros.h:

```
#ifndef __COLECCION_LIBROS__H
#define __COLECCION_LIBROS__H
#include "TablaHash.h"
#include "Arbol.h"
#include "ListaApariciones.h"
#include "Aparicion.h"
#include "Libro.h"
#include <iostream>
using namespace std;


class ColeccionLibros{
    private:
        TablaHash tablahash;
        Arbol arbol;
    public:
        ColeccionLibros();
        void insertarLibro(Libro * nuevo);
        Libro * consultarLibro(unsigned long long int isbn);
        void insertarAparicionPalabra(string cadena, Aparicion aparicion);
        ListaApariciones * obtenerAparicionesPalabra(string cadena);
        ListaApariciones * interseccionAparicionesDePalabras(string cadena);
        ListaApariciones * unionAparicionesDePalabras(string cadena);
};


#endif
```

Interprete.cpp:

```
#include "Interprete.h"
#include "ColeccionLibros.h"
#include "ListaApariciones.h"
#include "Aparicion.h"
#include "Libro.h"
#include "TextosFunciones.h"
#include <sstream>
#include <iostream>
using namespace std;


// Insertamos la información libro y su contenido en la colección de libros
void insertar(ColeccionLibros &dic){
    Libro * libro = new Libro();
    libro->leer();
          int capitulo_actual = 1;  // aparición
    int parrafo_actual = 1; // aparición
    int num_parrafos_totales = 1;   // información libro
    int num_palabras_totales = 0;   // información libro
    int num_FL = 0; // variable para controlar los párrafos
    string linea;
    string palabra;
    while( capitulo_actual <= libro->get_num_capitulos() ){
        getline(cin, linea);
        linea = normalizar(linea) + " FL";   // añadimos a la línea leída "FL", para así poder
contar los saltos de líneas facilmente
        istringstream palabras(linea);
        while(palabras >> palabra){
            if(palabra == "findecapitulo"){
```

```
            if( capitulo_actual != libro->get_num_capitulos() ) ++num_parrafos_totales;
            ++capitulo_actual;
            num_FL = 0;
            parrafo_actual = 1;
        }
        else{
            if(palabra == "FL"){
                ++num_FL;
                if(num_FL == 2){          // cuando los fin de línea son igual a 2 sumamos un párrafo tatno a la aparicón como al libro
                    ++num_parrafos_totales;
                    ++parrafo_actual;
                }
            }
            else{
                Aparicion aparicion = Aparicion(libro, capitulo_actual, parrafo_actual);
                dic.insertarAparicionPalabra(palabra, aparicion);
                        ++num_palabras_totales;
                        num_FL = 0;                  // si la palabra no es FL se queda en 0 e insrtamos la palabra y la aparición en la colección
            }
        }
    }
  }
  libro->mostrarLargo();
  cout << endl;
  cout << num_parrafos_totales  << " párrafos" << endl;
  cout << num_palabras_totales << " palabras" << endl;
  dic.insertarLibro(libro);       // finalmente, insertamos el libro
}

// Consulta si un libro se encuentra en una colección, si es así, muestra sus propiedades
void consultaISBN(ColeccionLibros &dic){
```

```
    unsigned long long int isbn;
    cin >> isbn;
          Libro * libro = dic.consultarLibro(isbn);
    if(libro == NULL) cout << "ISBN " << isbn << " no encontrado" << endl;
    else{
        cout << "ISBN " << isbn << " encontrado" << endl;
        libro->mostrarLargo();
        cout << endl;
    }

}

void consultaAND(ColeccionLibros &dic){
    string cadena;
    cin.ignore(1, ' ');
    getline(cin, cadena);
    cadena = normalizar(cadena);
    cout << "a " << cadena << endl;
    ListaApariciones * intersecionPalabras =
dic.interseccionAparicionesDePalabras(cadena);
    if( intersecionPalabras == NULL)
        cout << "Total: 0 resultados" << endl;
    else{
        intersecionPalabras->mostrar();
        delete intersecionPalabras; // liberamos la lista de la intersección que estaba en
memoria dinámica una vez la hemos mostrado
    }
}

void consultaOR(ColeccionLibros &dic){
    string cadena;
    cin.ignore(1, ' ');
    getline(cin, cadena);
    cadena = normalizar(cadena);
```

```
    cout << "o " << cadena << endl;
    ListaApariciones * unionPalabras = dic.unionAparicionesDePalabras(cadena); //
ESTO SERIA LO QUE TENIAMOS EN EL 302
    if( unionPalabras == NULL)
        cout << "Total: 0 resultados" << endl;
    else{
        unionPalabras->mostrar();
        delete unionPalabras; // liberamos la lista de la intersección que estaba en
memoria dinámica una vez la hemos mostrado
    }
}

void salir(){
}

// Se encarga de selección que acción realizar sobre una colección de libros
void interprete(char comando, ColeccionLibros &dic){
    if(comando == 'i') insertar(dic);
    else if(comando == 'c') consultaISBN(dic);
    else if(comando == 'a') consultaAND(dic);
    else if(comando == 'o') consultaOR(dic);
    else if(comando == 's') salir();
}
```

Interprete.h:

```
#ifndef __INTERPRETE_H__
#define __INTERPRETE_H__
#include "ColeccionLibros.h"

void insertar(ColeccionLibros &dic);
```

```
void consultaISBN(ColeccionLibros &dic);

void consultaAND(ColeccionLibros &dic);

void consultaOR(ColeccionLibros &dic);

void salir();

void interprete(char comando, ColeccionLibros &dic);

#endif
```

main.cpp:

```
#include "Interprete.h"
#include "ColeccionLibros.h"

ColeccionLibros dic;

int main(void){
    char comando;
    while(cin >> comando){
        interprete(comando, dic);
    }
}
```

TextosFunciones.cpp:

```
#include <iostream>
```

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


char mayusculas_a_minusculas(char c){
    if(c >= 'A' && c <= 'Z') return tolower(c);
    else return c;
}

char caracteres_especiales_a_normales(char c){
    if(c==char(0xA1))   return 'a';
    else if (c==char(0xA9))      return 'e';
    else if (c==char(0xAD))      return 'i';
    else if (c==char(0xB3))      return 'o';
    else if (c==char(0xBA)) return 'u';
    else if (c==char(0x9A)) return 'u';
    else if (c==char(0xBC)) return 'u';
    else if (c==char(0x81)) return 'a';
    else if (c==char(0x89)) return 'e';
    else if (c==char(0x8D)) return 'i';
    else if (c==char(0x93)) return 'o';
    else if (c==char(0x9C)) return 'u';
    else return ' ';
}

string normalizar(string input){
        string output;
        for(unsigned int i = 0; i < input.length(); ++i){
                if(input[i] >= 'a' && input[i] <= 'z')       output += input[i];
                else if(input[i] >= 'A' && input[i] <= 'Z')          output +=
mayusculas_a_minusculas(input[i]);
                else if(input[i] == char(0xC3)){
        ++i;
```

```
        if(input[i] == char(0x91) || input[i] == char(0xB1)) output = output + char(0xC3) + char(0xB1); // concatenacion necesaria para la ñ
        else    output += caracteres_especiales_a_normales(input[i]);
                }
                else output += ' ';
        }
        return output;
}
```

TextosFunciones.h:

```
#ifndef _TEXTOSFUNCIONES_H_
#define _TEXTOSFUNCIONES_H_
#include <iostream>
using namespace std;
char mayusculas_a_minusculas(char c);

char caracteres_especiales_a_normales(char c);

string normalizar(string input);

#endif
```

Makefile:

```
a.out: main.o Interprete.o ColeccionLibros.o TablaHash.o Arbol.o Nodo.o ListaApariciones.o Aparicion.o Libro.o TextosFunciones.o
        g++ -Wall main.o Aparicion.o Libro.o TextosFunciones.o Interprete.o ColeccionLibros.o TablaHash.o Nodo.o Arbol.o ListaApariciones.o
```

*main.o: main.cpp Interprete.h TextosFunciones.h Libro.h Aparicion.h ColeccionLibros.h ListaApariciones.h Arbol.h Nodo.h*

*g++ -c main.cpp*

*Interprete.o: Interprete.cpp Interprete.h TextosFunciones.h Libro.h Aparicion.h ColeccionLibros.h ListaApariciones.h Arbol.h Nodo.h*

*g++ -c Interprete.cpp*

*ColeccionLibros.o: ColeccionLibros.cpp ColeccionLibros.h TablaHash.h Arbol.h Aparicion.h Libro.h ListaApariciones.h*

*g++ -c ColeccionLibros.cpp*

*TablaHash.o: TablaHash.cpp TablaHash.h Libro.h*

*g++ -c TablaHash.cpp*

*Arbol.o: Arbol.cpp Arbol.h Nodo.h ListaApariciones.h Aparicion.h Libro.h*

*g++ -c Arbol.cpp*

*Nodo.o: Nodo.cpp Nodo.h Aparicion.h ListaApariciones.h Libro.h*

*g++ -c Nodo.cpp*

*ListaApariciones.o: ListaApariciones.cpp ListaApariciones.h Aparicion.h Libro.h*

*g++ -c ListaApariciones.cpp*

*Aparicion.o: Aparicion.cpp Aparicion.h Libro.h*

*g++ -c Aparicion.cpp*

*Libro.o: Libro.cpp Libro.h*

*g++ -c Libro.cpp*

*TextosFunciones.o: TextosFunciones.cpp TextosFunciones.h*

*g++ -c TextosFunciones.cpp*