Universidad Nacional Experimental De Guayana
Vicerrectorado Académico
Coordinación General De Pregrado
Proyecto De Carrera: Ingeniería En Informática
Estructuras De Datos – Sección 1

Proyecto 3 – Analizador De Artículos

Profesor:
Carlos Abaffy

Estudiantes:
Abiham Ramos 30.735.535
Inés Sánchez 30.898.804
Manuel Rodríguez 30.292.820

Puerto Ordaz, 03-03-2024

Análisis Del Problema

El propósito de este proyecto es diseñar e implementar un sistema para procesar el contenido de un artículo, proporcionando funcionalidades específicas para gestionar y analizar la información de manera eficiente. La arquitectura del sistema seguirá un enfoque de tres capas para garantizar modularidad y escalabilidad. Para la implementación de este sistema se tomarán en cuenta los siguientes factores:

Volumen De Datos:

- Debe poder manejar al rededor de 20.000 palabras si es requerido.
- Se espera que el volumen de datos aumente en el futuro.

Restricciones:

- Tiempo: Los algoritmos y estructuras de datos deben ser eficientes.
- Tecnología: C++.

Capa De Presentación:

La interfaz de usuario será la capa de presentación del sistema. Aquí, los usuarios interactuarán con las diversas funcionalidades a través de un menú intuitivo. Las opciones disponibles incluirán:

- **Leer y Procesar Archivo:** Permite cargar un archivo y procesar su contenido, identificando palabras, capítulos y páginas.

- **Imprimir Índice General:** Muestra el índice ordenado alfabéticamente, indicando las páginas en las que aparece cada palabra.

- **Imprimir Índice De Un Capítulo:** Permite seleccionar un capítulo específico y muestra su índice ordenado.

- **Búsqueda Parcial De Palabra:** Realiza una búsqueda parcial de una palabra, mostrando las páginas en las que aparece.

- **Eliminar Palabra Del Índice:** Permite eliminar una palabra específica del índice.

- **Imprimir Contenido Del Documento:** Muestra el contenido del documento, indicando en qué página inicia cada capítulo.

- **Información Estadística:** Proporciona el número total de capítulos, páginas, líneas y palabras (tanto totales como únicas).

Capa De Lógica De Negocio:

Esta capa es responsable de procesar la información y realizar las operaciones solicitadas. Aquí se implementarán las funciones para leer, analizar y gestionar el índice. Algunas funciones clave incluyen:

- **Procesar Índice:** Identifica palabras, capítulos y páginas, y guarda la información necesaria.

- **Generar Índice General:** Crea el índice general ordenado alfabéticamente.

- **Generar Índice De Capítulo:** Crea un índice específico para un capítulo dado.

- **Buscar Palabra:** Realiza una búsqueda de una palabra en el índice.

- **Eliminar Palabra:** Elimina una palabra específica del índice.

- **Estadísticas Del Documento:** Calcula el número total de capítulos, páginas, líneas y palabras.

Capa De Acceso a Datos:

La capa de acceso a datos se encargará de la lectura y escritura de información. Aquí se implementarán funciones para cargar y el archivo deseado y proporcionarlo a otras funciones para que la procesen

Este enfoque de tres capas proporcionará una estructura modular y organizada, facilitando el mantenimiento y la expansión del sistema. Además, se utilizará estructuras

de datos eficientes para manejar grandes volúmenes de palabras y optimizar tiempos de respuestas.

Elección De Las Estructuras De Datos

Para el guardado del índice, se implementó una tabla hash, debido a que nos proporcionaba los siguientes beneficios:

- **Eficiencia:** Las tablas hash ofrecen un acceso rápido a los datos, lo que es crucial para un sistema que debe procesar grandes cantidades de texto. En nuestro caso, la cantidad de palabras puede ser considerable, y una estructura de datos eficiente como una tabla hash permite realizar búsquedas y actualizaciones de forma rápida.

- **Escalabilidad:** Las tablas hash son estructuras de datos escalables, lo que significa que pueden adaptarse a grandes cantidades de datos sin perder eficiencia. Esto es importante para nuestro proyecto, ya que es posible que necesitemos procesar documentos cada vez más grandes.

En comparación con otras estructuras de datos:

- **Árboles:** Los árboles binarios de búsqueda también podrían usarse para almacenar el índice. Sin embargo, su tiempo de acceso promedio es O(log n), mientras que el de las tablas hash es O(1). Esto significa que las tablas hash son generalmente más rápidas para operaciones de búsqueda y actualización.

- **Grafos:** Los grafos no son una buena opción para almacenar el índice en este caso. Su estructura es más compleja que la de las tablas hash y no ofrecen las mismas ventajas en términos de eficiencia y manejo de colisiones.

Breve Explicación Uso De La Tabla Hash:

La tabla hash se tenía pensado para manejar el indice de palabras, el funcionamiento general de este iba a ser el siguiente, se iba a utilizar como clave de acceso a alguna

posición de la tabla (index) la palabra en sí que se va a procesar y dentro de ella se guardará los datos necesarios.

Consideraciones En El Uso De Las Tablas Hash:

Aunque anteriormente mostramos las razones por la cuál el uso de una tabla hash sería beneficioso para la realización de ciertas funciones del proyecto, también debemos de tener en cuenta que existen ciertas desventajas o consideraciones al usar las tablas hash que no podemos olvidar, como por ejemplo:

Paradoja Del Cumpleaños: en su forma más simple, el problema del cumpleaños o paradoja del cumpleaños es un fenómeno en las matemáticas y la informática que tiene implicaciones directas en las tablas hash. Este pregunta cuál es la probabilidad de que, en un grupo de personas seleccionadas al azar, al menos dos personas compartan el mismo cumpleaños. Llevado a nuestro proyecto y sabiendo que se toma las palabras como clave/key sería la probabilidad de que palabras completamente distintas de sí, compartan un mismo índice en la tabla. Por lo tanto es algo a tener en cuenta para por ejemplo, el ordenamiento alfabético y la búsqueda.

Manejo De Colisiones: en las tablas hash pueden ocurrir colisiones que deben de ser manejadas, en el contexto de nuestro proyecto, son situaciones en las que dos o más palabras tienen el mismo valor hash. En nuestro caso, es probable que haya palabras que se repitan en el documento, pero en diferentes capítulos. La tabla hash se encarga de gestionar estas colisiones de manera eficiente, evitando que afecten al rendimiento del sistema.

En nuestro caso usamos la ***técnica de encadenamiento*** para manejar las colisiones la cuál consiste en en pocas palabras que todos los elementos cuyas claves generan el mismo indice primario, se encadenan mediante una lista externa en la tabla

Búsqueda Parcial – Algoritmo De Boyer-Moore

Para la búsqueda parcial de palabras utilizamos el algoritmo de Boyer-Moore, el cuál es un algoritmo de búsqueda parcial que funciona como un método eficiente para buscar

una cadena dentro de otra. A diferencia de algunos algoritmos que procesan la cadena en la que se busca, este algoritmo procesa la cadena objetivo (la que se está buscando).. Su tiempo de ejecución es lineal en el tamaño de la cadena buscada.

Identificación De Palabras

Como sabemos que el programa debe de ser capaz de analizar el texto y identificar, guardar y contabilizar las palabras además de otras “etiquetas guías” las cuáles estarán en el artículo y son necesarias para el correcto y certero procesamiento del texto, el programa debe de ser capaz de procesar el texto de forma correcta. Las palabras serán todo el conjunto de caracteres que se encuentra separado de otro (espaciado), posteriormente este será analizado y filtrado.

Etiquetas Guías:
Las etiquetas guías serán etiquetas especiales que no son parte del texto sino que ayudan a saber información deseable. Las etiquetas son las siguientes:

- **Identificador De Capitulo:** para poder saber en qué capitulo se encuentra el programa a la hora de estar analizando el texto y así poder relacionar las palabras con un capítulo.

```
Estructura : <capitulo xx – nombre>
Ejemplo:
<capitulo 2 – Python The Hard Way>
python is not hard… really.
```

- **Identificador De Página:** para poder saber la página actual o recurrente de la que andamos analizando información usamos una estructura especifica para declarar el inicio de una página.

```
Estructura: <pagina xxx>
Ejemplo:
<pagina 003>
Hello world!
```

Claramente estas etiquetas se pueden usar de manera combinada la recomendación de su uso en conjunto simplemente es el siguiente: por temas de análisis de información

del documento es necesario primero la declaración del inicio o número de una página antes que de un capítulo, lo que quiere decir que:

Incorrecto:

```
<pagina 003>
<capitulo 2 – Python The Hard Way>
Hello World!
```

Correcto:

```
<capitulo 2 – Python The Hard Way>
<pagina 003>
Hello World!
```

Filtro De Palabras: ahora vamos a mencionar qué y qué no consideraremos una palabra además de los casos especiales y/o excepcionales

- **Números o combinados:** debido al requerimiento de un ordenamiento alfabético todo número o mezcla de número y letras será filtrado. Ejemplo: 231, hol3
- **Caracteres Especiales:** claramente no consideraremos caracteres especiales como una palabra, ya sea de forma individual o junto alguna letra. Ejemplo: *, /%, hol*+

Casos Especiales Caracteres Especiales:

Existen signos que se podrían considerar como caracteres especiales pero tienen un uso gramatical en muchos casos y deben de ser considerados por ejemplo:

- **Paréntesis:** los paréntesis serán obviados y extraeremos el contenido dentro de este sí y solo sí están bien colocadas:

  Correcto:
  (un mundo inimaginable)

  Incorrecto:
  un(mund)o inimaginable

- **Signos De Exclamación o Pregunta:** estos signos serán obviados y solo se extraerá la palabra textual en sí, sí y solo sí están bien colocados. Ejemplo:

Correcto:
¿Estás Bien?

Incorrecto:
Está?s Bi¿en

- **Signos De Acentuación, Separación, etc:** estos signos que mencionamos tales como “, . ` : “ serán obviados sí y solo sí se encuentran en donde gramaticalmente se supone que se encuentren, como el caso anterior.

  Correcto:
  Hola, mira un ejemplo:

  Incorrecto:
  Ho,la : mira un ejem.plo

Casos Excepcionales: por último los enlistamientos del tipo a), b), c) … serán completamente ignorados, al igual que todo lo que tenga parecido con alguna URL

Algoritmo General

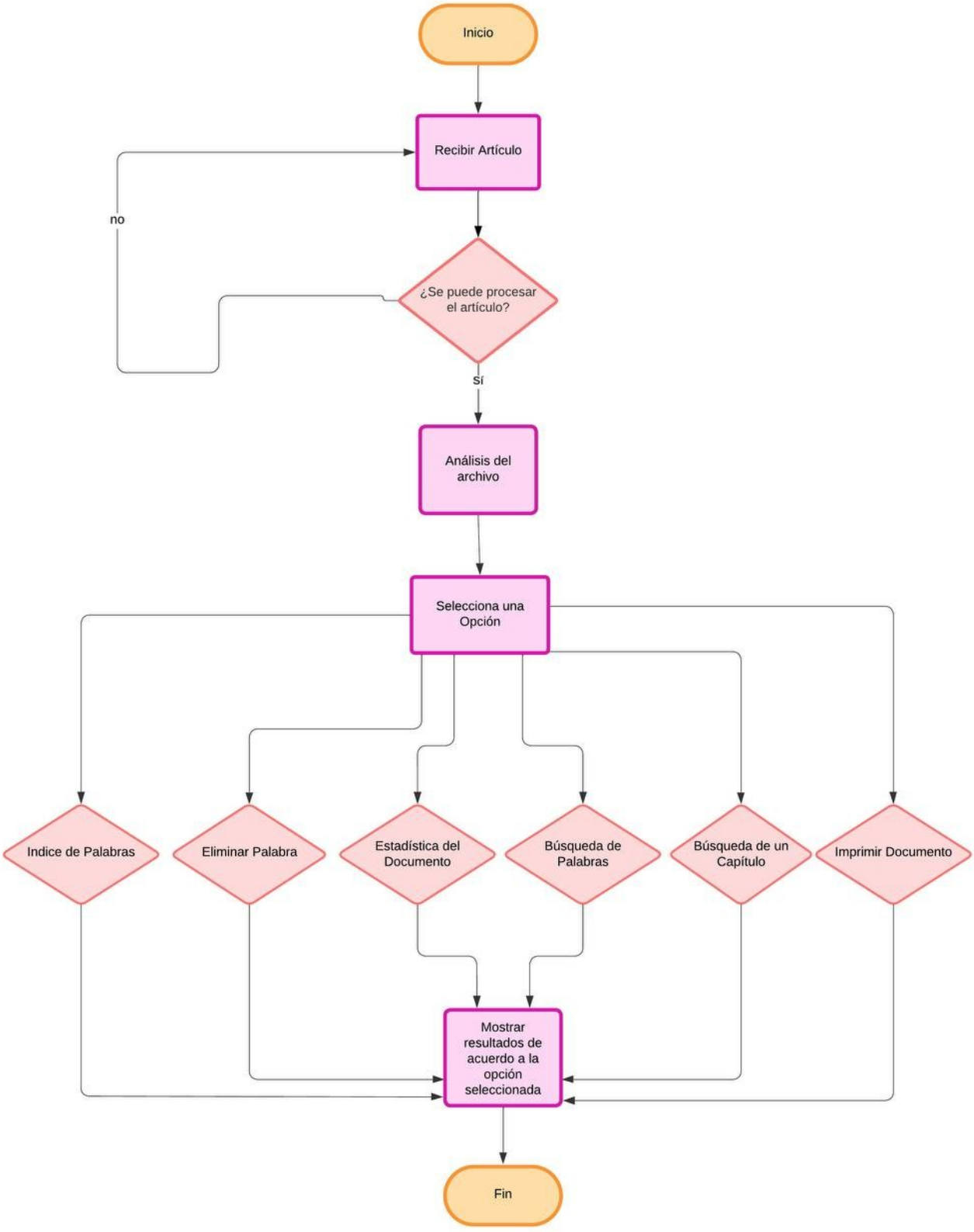
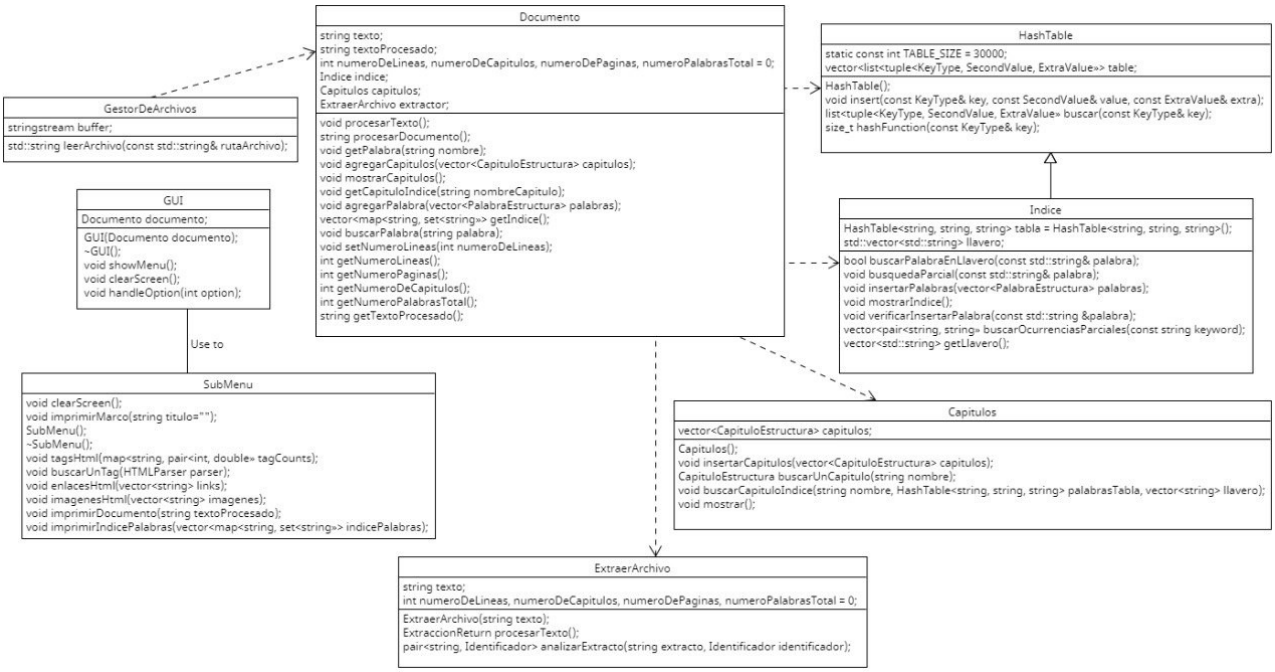


Diagrama De Clases UML



Casos De Prueba:

1. **Leer y Procesar Archivo:**

Entrada: Archivo de texto con contenido variado, incluyendo palabras, capítulos y páginas.

Salida Esperada: Confirmación de procesamiento exitoso y almacenamiento de información.

2. **Imprimir Índice General:**

Entrada: Archivo procesado con palabras en diversas páginas.

Salida Esperada: Índice general ordenado alfabéticamente con páginas correspondientes.

3. **Imprimir Índice de un Capítulo:**

Salida Esperada: Índice específico para un capítulo seleccionado.

4. **Búsqueda Parcial de Palabra:**

Entrada: Palabra parcial para buscar en el índice.

Salida Esperada: Listado de páginas donde aparece la palabra parcial y las páginas en dónde aparecen.

5. **Eliminar Palabra del Índice:**

Entrada: Palabra a eliminar del índice.

Salida Esperada: Confirmación de eliminación y actualización del índice.

6. **Imprimir Contenido del Documento:**

Entrada: Archivo procesado con capítulos y páginas identificadas.

Salida Esperada: Mostrar el contenido del documento con inicio de capítulos en cada página.

7. **Información Estadística:**

Entrada: Archivo procesado con información variada.

Salida Esperada: Números totales y únicos de capítulos, páginas, líneas y palabras.

Código Del Proyecto

main.cpp

```
#include <iostream>
#include <ncurses.h>
#include <ostream>
#include "./datos/GestorDeArchivos.hpp"
#include "./presentacion/GUI.h"
#include "./negocio/Documento.hpp"
#include "./presentacion/GUI.h"

using namespace std;

int main (int argc, char *argv[]) {

if (argc < 2) {
cerr << "Uso: " << argv[0] << " <file_path>\n";
return 1;
}

string articulo = GestorDeArchivos().leerArchivo(argv[1]);
if (articulo == "") {
cerr << "No se pudo leer el archivo: " << argv[1] << std::endl;
return 1;
}
Documento documento = Documento(articulo);
documento.procesarTexto();
documento.procesarDocumento();

GUI gui(documento);
gui.showMenu();

return 0;
}
```

GUI.h

```
#ifndef GUI_H
#define GUI_H

//#include <ncurses.h>
#include <iostream>
#include <vector>
#include <string>
#include "../negocio/Documento.hpp"

/**
* La clase GUI representa una interfaz gráfica de usuario.
* Contiene métodos para mostrar un menú, limpiar la pantalla y manejar las
opciones seleccionadas por el usuario.
*/
class GUI {

private:
```

```
Documento documento;

public:
GUI(Documento& documento);
~GUI();
void showMenu();
void clearScreen();

private:
void handleOption(int option);
};

#endif // GUI_H
```

GUI.cpp

```
#include <ncurses.h>
#include "GUI.h"
#include "SubMenu.h"

GUI::GUI(Documento& documento) : documento(documento) {

initscr();
start_color(); // Habilitar el uso de colores
keypad(stdscr, TRUE); // Habilitar el teclado numérico
init_pair(1, COLOR_WHITE, COLOR_BLACK);
init_pair(2, COLOR_BLACK, COLOR_WHITE);
init_pair(3, COLOR_BLUE, COLOR_BLACK);

showMenu();
}

GUI::~GUI() {
endwin(); // Finalizar ncurses al destruir la instancia de la clase
}

void GUI::clearScreen() {
clear();
refresh();
}

void GUI::showMenu() {

int max_y, max_x;
int current_option = 1;
getmaxyx(stdscr, max_y, max_x); // Obtener el tamaño de la terminal

std::vector<std::string> ascii_art = {
" ,--,--'  .  ,.  .  ",
"`- |  ,-.  .  ,  |- /  |  ,-.  ,-.  |  .  .  ,_,  ,-.  ,-.  ",
" ,  |  |-'  X  |  /~~|-.  |  |  ,-|  |  |  |  /  |-'  |  ",
" `-'  `-'  '  `  `'  ,'  `-'  '  '  `-^  `'  `-|  '\"'  `-'  '  ",
" /|  ",
" `-'  "
};

do {
clearScreen();
// Calcular las coordenadas para centrar el menú
int menu_y = (max_y - 11) / 2;
int menu_x = (max_x - 40) / 2;

attron(COLOR_PAIR(1));
// Imprimir el arte ASCII
```

```
for (size_t i = 0; i < ascii_art.size(); ++i) {
mvprintw(menu_y - 7 + i, menu_x - 5, ascii_art[i].c_str());
}
attroff(COLOR_PAIR(1));

attron(COLOR_PAIR(3));
mvprintw(menu_y - 1, menu_x - 1, "+---------------------------------------+");
mvprintw(menu_y + 10, menu_x - 1, "+---------------------------------------+");
for (int i = 0; i < 11; ++i) {
mvprintw(menu_y + i, menu_x - 1, "*");
mvprintw(menu_y + i, menu_x + 40, "*");
}
attroff(COLOR_PAIR(3));

// Imprimir el menú
attron(COLOR_PAIR(3));
mvprintw(menu_y + 1, menu_x, "1. Indice De Palabras");
mvprintw(menu_y + 2, menu_x, "2. Eliminar Palabra");
mvprintw(menu_y + 3, menu_x, "3. Estadistica Del Documento");
mvprintw(menu_y + 4, menu_x, "4. Busqueda De Palabras");
mvprintw(menu_y + 5, menu_x, "5. Busqueda De Un Capitulo ");
mvprintw(menu_y + 6, menu_x, "6. Imprimir Documento");
attroff(COLOR_PAIR(3));

// Pintar opción actual
attron(COLOR_PAIR(2));
switch (current_option) {
case 1:
mvprintw(menu_y + 1, menu_x, "1. Indice De Palabras");
break;
case 2:
mvprintw(menu_y + 2, menu_x, "2. Eliminar Palabra");
break;
case 3:
mvprintw(menu_y + 3, menu_x, "3. Estadistica Del Documento");
break;
case 4:
mvprintw(menu_y + 4, menu_x, "4. Busqueda De Palabras");
break;
case 5:
mvprintw(menu_y + 5, menu_x, "5. Busqueda De Un Capitulo ");
break;
case 6:
mvprintw(menu_y + 6, menu_x, "6. Imprimir Documento ");
break;
}
attroff(COLOR_PAIR(2));

mvprintw(menu_y + 8, menu_x, "Seleccione una opción (w/s/enter) ");
refresh();
int ch = getch();
if (ch == 'w' || ch == KEY_UP) {
if (current_option > 1) {
current_option--;
}
} else if (ch == 's' || ch == KEY_DOWN) {
if (current_option < 6) {
current_option++;
}
} else if (ch == '\n') {
if (current_option >= 1 && current_option <= 6) {
handleOption(current_option);
}
```

```
}
} while (true);
}

void GUI::handleOption(int option) {
SubMenu submenu;

switch (option) {
case 1:
submenu.imprimirIndicePalabras(documento.getIndice());
break;
case 2:
submenu.EliminarPalabra(documento);
break;
case 3:
submenu.imprimirEstadisticas(documento.getNumeroDeCapitulos(),
documento.getNumeroLineas(),
documento.getNumeroPaginas(), documento.getNumeroPalabrasTotal(),
documento.getNumeroPalabrasUnicas());
break;
case 4:
submenu.MostrarPalabras(documento);
break;

case 5: {
submenu.buscarUnCapitulo(documento);
break;
case 6:
clearScreen();
submenu.imprimirDocumento(documento.getTextoProcesado());
refresh();
break;
}
}
}
```

SubMenu.h

```
// SubMenu.h
#ifndef SubMenu_h
#define SubMenu_h

#include <iostream>
#include <map>
#include <ncurses.h>
#include <set>
#include <vector>
#include "../negocio/Documento.hpp"

using namespace std;

class SubMenu {

private:
void clearScreen();
void imprimirMarco(string titulo="");

public:
SubMenu(); // constructor
~SubMenu(); // destructor

void imprimirDocumento(string textoProcesado);
void imprimirIndicePalabras(vector<map<string, set<string>>> indicePalabras);
void buscarUnCapitulo(Documento documento);
```

```
void imprimirConScroll(const vector<string>& lines);
void imprimirEstadisticas(int numeroDeCapitulos, int numeroDeLineas, int
numeroDePaginas, int numeroPalabrasTotal, int numeroDePalabrasUnicas);
void EliminarPalabra(Documento& documento);
void MostrarPalabras(Documento documento);
};

#endif
```

SubMenu.cpp

```
// SubMenu.cpp
#include "SubMenu.h"
#include <array>
#include <cctype>
#include <cstring>
#include <ncurses.h>
#include <sstream>
#include <algorithm>

#define SCROLL_LINE_STEP 1
#define MARGIN 2

SubMenu::SubMenu() {
initscr();
start_color();
init_pair(1, COLOR_WHITE, COLOR_BLACK);
init_pair(2, COLOR_BLACK, COLOR_WHITE);
init_pair(3, COLOR_BLUE, COLOR_BLACK);
}

SubMenu::~SubMenu() { endwin(); }

void SubMenu::clearScreen() {
clear();
refresh();
}

void SubMenu::imprimirMarco(string titulo) {
int terminal_width = COLS; // Ancho de la terminal
int terminal_height = LINES; // Altura de la terminal

clearScreen();
attron(COLOR_PAIR(3));
// Top border
mvprintw(0, 0, "+");
for (int i = 1; i < terminal_width - 1; ++i) {
mvprintw(0, i, "-");
}
mvprintw(0, terminal_width - 1, "+");
// Side borders
for (int i = 1; i < terminal_height - 1; ++i) {
mvprintw(i, 0, "|");
mvprintw(i, terminal_width - 1, "|");
}
// Bottom border
mvprintw(terminal_height - 1, 0, "+");
for (int i = 1; i < terminal_width - 1; ++i) {
mvprintw(terminal_height - 1, i, "-");
}
mvprintw(terminal_height - 1, terminal_width - 1, "+");
// Title
mvprintw(1, (COLS - strlen(titulo.c_str())) / 2, titulo.c_str());
attroff(COLOR_PAIR(1));
```

```
refresh();
}

void addLineWithWrapping(std::vector<std::string>& lines, const std::string&
line, size_t maxLineWidth) {
size_t start = 0;
while (start < line.length()) {
size_t end = start + maxLineWidth;
if (end > line.length()) end = line.length();

// Encuentra el último espacio en blanco para evitar cortar palabras
if (end < line.length()) {
size_t lastSpace = line.rfind(' ', end);
if (lastSpace != std::string::npos && lastSpace > start) {
end = lastSpace;
}
}

lines.push_back(line.substr(start, end - start));
start = end + ((end < line.length()) ? 1 : 0); // Saltar el espacio si no es el
final de la línea
}
}

void SubMenu::imprimirDocumento(string textoProcesado) {
// Convertir el texto procesado en un vector de strings, cada uno representando
una línea
std::istringstream iss(textoProcesado);
std::string line;
std::vector<std::string> lines;
while (getline(iss, line)) {
// Verifica si la línea se ajusta dentro del ancho del marco
int maxLineWidth = COLS - 2 * MARGIN; // Ajuste según el ancho del marco
if (line.length() <= maxLineWidth) {
lines.push_back(line);
} else {
// Si la línea es más larga, divídela en sublíneas
for (size_t i = 0; i < line.length(); i += maxLineWidth) {
lines.push_back(line.substr(i, maxLineWidth));
}
}
}
// Inicialización para capturar eventos del mouse
mousemask(ALL_MOUSE_EVENTS | REPORT_MOUSE_POSITION, NULL);
keypad(stdscr, TRUE);
MEVENT event;

int scroll_offset = 0; // Esto va a llevar el registro de cuánto hemos
scrolleado

// Bucle principal para mostrar el documento y manejar el scroll
bool quit = false;
while (!quit) {
clearScreen(); // Limpia la pantalla antes de volver a dibujar
imprimirMarco("Titulo del Documento"); // Asegúrate de que este método no limpie
la pantalla

// Imprimir el documento desde el scroll_offset
int max_line = LINES - 3; // Ajuste para el espacio vertical del marco
for (int i = 0; i < max_line && (i + scroll_offset) < lines.size(); ++i) {
mvprintw(i + 2, MARGIN, "%s", lines[i + scroll_offset].c_str()); // Ajusta para
el margen horizontal
}
// Capturar eventos
```

```
int ch = getch();
switch (ch) {
case KEY_MOUSE:
if (getmouse(&event) == OK) {
if (event.bstate & BUTTON4_PRESSED) { // Scroll hacia arriba
scroll_offset -= SCROLL_LINE_STEP;
if (scroll_offset < 0) scroll_offset = 0;
} else if (event.bstate & BUTTON5_PRESSED) { // Scroll hacia abajo
scroll_offset += SCROLL_LINE_STEP;
if (scroll_offset > lines.size() - max_line) scroll_offset = lines.size() -
max_line;
}
}
break;
case 'q': // Presionar 'q' para salir
quit = true;
break;
// Añade más casos si necesitas más controles
}
}
}

void SubMenu::imprimirIndicePalabras(vector<map<string, set<string>>>
indicePalabras) {
map <string, set<string>> palabrasOrdenadas;

for (const auto& mapa : indicePalabras) {
for (const auto& par : mapa) {
string palabra = par.first;
set<string> informacion = par.second;
palabrasOrdenadas[palabra] = informacion;
}
}

mousemask(ALL_MOUSE_EVENTS | REPORT_MOUSE_POSITION, NULL);
keypad(stdscr, TRUE);
MEVENT event;

int scroll_offset = 0;
bool quit = false;

while (!quit) {
clearScreen();
imprimirMarco("Índice de Palabras");

int max_line = LINES - 3;
int current_line = 2;

int i = 0;
for (const auto& par : palabrasOrdenadas) {
if (i < scroll_offset) {
i++;
continue;
}

// Cada palabra y su información asociada
string palabra = par.first;
set<string> informacion = par.second;

mvprintw(current_line++, MARGIN, "Palabra: %s", palabra.c_str());

for (const auto& info : informacion) {
if (current_line <= max_line) {
mvprintw(current_line++, MARGIN + 2, "Info: %s", info.c_str());
```

```
} else {
break; // Rompe el bucle si alcanzamos el límite de líneas
}
}

if (current_line <= max_line) {
// Separador entre palabras
mvprintw(current_line++, MARGIN, "----------------");
} else {
break; // Rompe el bucle si alcanzamos el límite de líneas
}
}

int ch = getch();
switch (ch) {
case KEY_MOUSE:
if (getmouse(&event) == OK) {
if (event.bstate & BUTTON4_PRESSED) {
scroll_offset -= SCROLL_LINE_STEP;
if (scroll_offset < 0) scroll_offset = 0;
} else if (event.bstate & BUTTON5_PRESSED) {
scroll_offset += SCROLL_LINE_STEP;
int max_offset = static_cast<int>(indicePalabras.size()) - max_line;
if (scroll_offset > max_offset) scroll_offset = max_offset;
}
}
break;
case 'q':
quit = true;
break;
}
}
}

void SubMenu::imprimirEstadisticas(int numeroDeCapitulos, int numeroDeLineas,
int numeroDePaginas, int numeroPalabrasTotal, int numeroDePalabrasUnicas) {
clearScreen();
box(stdscr, 0, 0);
int center = COLS / 2 - 9;
mvprintw(0, center, "ESTADISTICAS");

mvprintw(2, 2, "Numero de capitulos: %d", numeroDeCapitulos);
mvprintw(3, 2, "Numero de lineas: %d", numeroDeLineas);
mvprintw(4, 2, "Numero de paginas: %d", numeroDePaginas);
mvprintw(5, 2, "Numero de palabras totales: %d", numeroPalabrasTotal);
mvprintw(6, 2, "Numero de palabras unicas: %d", numeroDePalabrasUnicas);
getch();
}

void SubMenu::buscarUnCapitulo(Documento documento) {
imprimirMarco(""); // Imprime el marco inicial
attron(COLOR_PAIR(1));
echo(); // Habilita el eco de los caracteres ingresados por el usuario
mvprintw(LINES / 2 - 10, COLS / 2 - 10, "BUSCAR UN CAPITULO");
mvprintw(LINES / 2 - 10 + 2, COLS / 2 - 15, ">>> ");
attroff(COLOR_PAIR(1));

char userInput[256];
getnstr(userInput, sizeof(userInput) - 1);
noecho(); // Deshabilita el eco de los caracteres

string nombreCapitulo = userInput;
vector<string> resultado = documento.getCapituloIndice(nombreCapitulo);
```

```
imprimirConScroll(resultado);
}

void SubMenu::imprimirConScroll(const vector<string>& lines) {
mousemask(ALL_MOUSE_EVENTS | REPORT_MOUSE_POSITION, NULL);
keypad(stdscr, TRUE);
MEVENT event;

int scroll_offset = 0; // Lleva el registro de cuánto hemos scrolleado
int max_line = LINES - 3; // Ajuste para el espacio vertical del marco

// Bucle principal para mostrar el contenido y manejar el scroll
bool quit = false;
while (!quit) {
clearScreen(); // Limpia la pantalla antes de volver a dibujar
imprimirMarco("Resultado de la busqueda");

// Imprimir el contenido desde scroll_offset
int max_line_actual = min(max_line, static_cast<int>(lines.size() -
scroll_offset));
for (int i = 0; i < max_line_actual; ++i) {
mvprintw(i + 2, MARGIN, "%s", lines[i + scroll_offset].c_str()); // Ajusta para
el margen horizontal
}

// Capturar eventos
int ch = getch();
switch (ch) {
case KEY_MOUSE:
if (getmouse(&event) == OK) {
if (event.bstate & BUTTON4_PRESSED) { // Scroll hacia arriba
scroll_offset -= SCROLL_LINE_STEP;
if (scroll_offset < 0) scroll_offset = 0;
} else if (event.bstate & BUTTON5_PRESSED) { // Scroll hacia abajo
scroll_offset += SCROLL_LINE_STEP;
if (scroll_offset > lines.size() - max_line) scroll_offset = lines.size() -
max_line;
}
}
break;
case 'q': // Presionar 'q' para salir
quit = true;
break;
// Agrega más casos si necesitas más controles
}
}
}

void SubMenu::EliminarPalabra(Documento& documento) {
int fila = 3;
int columna = (COLS - 60) / 2;
int indice = 0;
int pagina = 0;
int elementosPorPagina = LINES - 10; // Calcula cuántos elementos caben en una
página

imprimirMarco("");
attron(COLOR_PAIR(1));
echo(); // Habilitar el eco de los caracteres ingresados por el usuario
mvprintw(LINES / 2 - 10, COLS / 2 - 10, "BUSCAR PALABRA");
mvprintw(LINES / 2 - 10 + 2, COLS / 2 - 15, ">>> ");
attroff(COLOR_PAIR(1));

char userInput[256];
```

```
getnstr(userInput, sizeof(userInput) - 1);
noecho();

// Llama al método eliminarPalabra de la clase Documento
bool palabraEliminada = documento.eliminarPalabra(userInput);

if (palabraEliminada) {
// Realiza las acciones necesarias después de eliminar la palabra
// Puedes mostrar un mensaje de éxito o realizar otras operaciones
mvprintw(LINES / 2, COLS / 2 - 10, "Palabra eliminada con éxito");
} else {
// La palabra no se encontró o no se pudo eliminar
mvprintw(LINES / 2, COLS / 2 - 10, "Palabra no encontrada o no se pudo
eliminar");
}

// Puedes agregar más lógica según tus necesidades
}

void SubMenu::MostrarPalabras(Documento documento) {
int fila = 3;
int columna = (COLS - 60) / 2;
int indice = 0;
int pagina = 0;
int elementosPorPagina = LINES - 10; // Calcula cuántos elementos caben en una
página

imprimirMarco("");
attron(COLOR_PAIR(1));
echo(); // Habilitar el eco de los caracteres ingresados por el usuario
mvprintw(LINES / 2 - 10, COLS / 2 - 10, "BUSCAR PALABRA");
mvprintw(LINES / 2 - 10 + 2, COLS / 2 - 15, ">>> ");
attroff(COLOR_PAIR(1));

char userInput[256];
getnstr(userInput, sizeof(userInput) - 1);
noecho(); // Deshabilitar el eco

std::string palabra(userInput);
auto resultados = documento.buscarPalabra(palabra);

std::vector<std::string> lines;
for (const auto& par : resultados) {
std::string linea = par.first + ": ";
for (const auto& ubicacion : par.second) {
linea += ubicacion + " ";
}
lines.push_back(linea);
}

imprimirConScroll(lines);
}
```

Indice.h

```
#ifndef INDICE_H
#define INDICE_H

#include <iostream>
#include <vector>
#include <algorithm>
#include "HashTable.h"
#include "ExtraerArchivo.hpp"
```

```
using namespace std;

class Indice {

public:
HashTable<string, string, string> tabla = HashTable<string, string, string>();

private:
std::vector<std::string> llavero;

private:
bool buscarPalabraEnLlavero(const std::string& palabra);

public:

std::map<std::string, std::set<std::string>> busquedaParcial(const std::string&
palabra);
void insertarPalabras(vector<PalabraEstructura> palabras);
void mostrarIndice();
void verificarInsertarPalabra(const std::string &palabra);
vector<std::string> getLlavero();
pair <bool, int> eliminarPalabraIndice(const string &palabra);
};

#endif // INDICE_H
```

Indice.cpp

```
#include <algorithm>
#include <iostream>
#include <vector>
#include "HashTable.h"
#include "Indice.h"

// Función para buscar una palabra en el vector llavero
bool Indice::buscarPalabraEnLlavero(const std::string &palabra) {
return std::find(llavero.begin(), llavero.end(), palabra) != llavero.end();
}

void Indice::insertarPalabras(vector<PalabraEstructura> palabras) {
for (const auto &palabra : palabras) {
verificarInsertarPalabra(palabra.palabra);
tabla.insert(palabra.palabra, palabra.pagina, palabra.capitulo);
}
}

// Verificar palabra e insertar en el atributo llavero
void Indice::verificarInsertarPalabra(const std::string &palabra) {
if (!buscarPalabraEnLlavero(palabra)) {
llavero.push_back(palabra);
std::sort(llavero.begin(), llavero.end()); // Ordenar el llavero alfabéticamente
}
}

// Función para eliminar coincidencias parciales de una palabra en la tabla
hashing
pair <bool, int> Indice::eliminarPalabraIndice(const string &palabra) {
int posicion = tabla.hashFunction(palabra);
list<tuple<string, string, string>> lista = tabla.table[posicion];
bool seElimino = false;
int numEliminados = 0;

for (auto it = lista.begin(); it != lista.end(); ) {
```

```
if ( std::search(std::get<0>(*it).begin(), std::get<0>(*it).end(),
palabra.begin(), palabra.end()) != std::get<0>(*it).end() ) {
it = lista.erase(it);
seElimino = true;
numEliminados++;
} else {
++it;
}
}
if (seElimino) {
tabla.table[posicion] = lista;
// eliminamos la palabra del llavero
llavero.erase(std::remove(llavero.begin(), llavero.end(), palabra),
llavero.end());
return {true, numEliminados};
}
return {false, 0};
}

std::map<std::string, std::set<std::string>> Indice::busquedaParcial(const
std::string& palabra) {
std::vector<std::string> llavesCoincidentes;

for (const std::string& llave : llavero) {
if (llave.find(palabra) != std::string::npos) {
llavesCoincidentes.push_back(llave);
}
}

std::map<std::string, std::set<std::string>> uniqueWordsAndPages; // Store
unique words and pages

// Recorrer la tabla hash y entrar dentro del listado y checar conincidencias
for (const std::string& llave : llavesCoincidentes) {
list<tuple<std::string, std::string, std::string>> palabras =
tabla.buscar(llave);
for (const auto& tupla : palabras) {
if (std::search(std::get<0>(tupla).begin(), std::get<0>(tupla).end(),
palabra.begin(), palabra.end()) != std::get<0>(tupla).end()) {
uniqueWordsAndPages[std::get<0>(tupla)].insert(std::get<1>(tupla)); // agregar
palabra y pagina al map
}
}
}

return uniqueWordsAndPages;
}

vector <string> Indice::getLlavero() {
return llavero;
}
```

HashTable.h

```
#ifndef HASHTABLE_H
#define HASHTABLE_H

#include <iostream>
#include <vector>
#include <list>
#include <tuple>
#include <functional> // Para std::hash
#include <map>
```

```
#include <set>

using namespace std;

template<typename KeyType, typename SecondValue, typename ExtraValue>
class HashTable {
private:
static const int TABLE_SIZE = 30000;
public:
vector<list<tuple<KeyType, SecondValue, ExtraValue>>> table;
public:
HashTable() {
table.resize(TABLE_SIZE);
}
void insert(const KeyType& key, const SecondValue& value, const ExtraValue&
extra) {
int index = hashFunction(key);
table[index].push_back(make_tuple(key, value, extra));
}
// Función para buscar un elemento por clave y devolver la lista completa
list<tuple<KeyType, SecondValue, ExtraValue>> buscar(const KeyType& key) {
int index = hashFunction(key);
return table[index];
}

size_t hashFunction(const KeyType& key) {
return std::hash<KeyType>{}(key) % TABLE_SIZE;
}

};

#endif // HASHTABLE_H
```

ExraerArchivo.hpp

```
#ifndef EXTRAER_ARCHIVO_H
#define EXTRAER_ARCHIVO_H

#include <iostream>
#include <vector>
using namespace std;

// Ayuda a identificar el tipo de procesamiento que se le dará a un texto
enum Identificador {
PALABRA,
PAGINA,
CAPITULO,
DESCONOCIDO,
IGNORAR // no representa una palabra válida
};

struct CapituloEstructura{
string nombre;
string paginaInicio;
};

struct PalabraEstructura{
string palabra;
string pagina;
string capitulo;
};

struct Contabilizados {
```

```
int numeroDeLineas;
int numeroDeCapitulos;
int numeroDePaginas;
int numeroDePalabrasTotal;
};

struct ExtraccionReturn {
vector<PalabraEstructura> palabras;
vector<CapituloEstructura> capitulos;
Contabilizados contabilizados;
};

class ExtraerArchivo {

private:
string texto;
int numeroDeLineas, numeroDeCapitulos, numeroDePaginas, numeroPalabrasTotal = 0;
public:
ExtraerArchivo(string texto);
ExtraccionReturn procesarTexto();
pair<string, Identificador> analizarExtracto(string extracto, Identificador
identificador);
};

#endif
```

ExtraerArchivo.cpp

```
#include "ExtraerArchivo.hpp"

#include <cctype>
#include <codecvt>
#include <iostream>
#include <locale>
#include <set>
#include <sstream>
#include <algorithm>

// Para manejar string <-> wstring para temas de utf-8
wstring_convert<std::codecvt_utf8<wchar_t>, wchar_t> converter;

ExtraerArchivo::ExtraerArchivo(string texto) {
this -> texto = texto;
}

pair < string, Identificador > ExtraerArchivo::analizarExtracto(string extracto,
Identificador identificador) {
string texto;

if (identificador == PAGINA || identificador == CAPITULO) {
size_t inicio = extracto.find(' ') + 1;
size_t fin = extracto.find('>', inicio);

if (inicio != string::npos && fin != string::npos) {
texto = extracto.substr(inicio, fin - inicio);
Identificador id = identificador == PAGINA ? PAGINA : CAPITULO;
return {texto, id};
}
return {extracto, DESCONOCIDO};
}

if (identificador == PALABRA) {
if (extracto.empty()) return {extracto,IGNORAR};
```

```
wstring acentos = L"áéíóúÁÉÍÓÚüÜñÑ";
set<wchar_t> cEspecialesValidosApertura = {L'¡', L'¿',L'('};
set<wchar_t> cEspecialesValidosCierre = {L'!', L'?', L')'};

wstring w_extracto = converter.from_bytes(extracto);
bool esValido = true;

// Verificar si el texto contiene caracteres no validos
for (int i = 0; i < w_extracto.size() && esValido; i++) {
wchar_t c = w_extracto[i];
bool esCaracterValido = cEspecialesValidosApertura.count(c) ||
cEspecialesValidosCierre.count(c) || c == L',' || c == L'.' || c == L':';
if (!iswalpha(c) && !esCaracterValido && acentos.find(c) == wstring::npos) {
esValido = false;
break;
}

// Verificar si . , : están en medio de la palabra
if ((c == L',' || c == L'.' || c == L':') && i != w_extracto.size() - 1) {
esValido = false;
break;
}
}

if (esValido) {
wchar_t ultimoCaracter = w_extracto[w_extracto.size() - 1];

// Remover los . , :
if (ultimoCaracter == L',' || ultimoCaracter == L'.' || ultimoCaracter ==L':') {
w_extracto = w_extracto.substr(0, w_extracto.size() - 1);
if (w_extracto.empty()) esValido = false;
}

// Verificar si es un enlistamiento tipo a) b) c)...
if (ultimoCaracter == L')' && esValido) {
wstring antesParentesis = w_extracto.substr(0, w_extracto.size()-1);

if (!all_of(antesParentesis.begin(), antesParentesis.end(), ::isalpha) ||
antesParentesis.size() < 2) {
esValido = false;
}
}

// Manejar los carácteres válidos con apertura y cierre
if (esValido) {
for (wchar_t c : cEspecialesValidosApertura) {
size_t posApertura = w_extracto.find(c);
if (posApertura != wstring::npos) {
if (posApertura != 0) {
esValido = false;
}
w_extracto = w_extracto.substr(1);
}
}
}
if (esValido) {
for (wchar_t c : cEspecialesValidosCierre) {
size_t posCierre = w_extracto.find(c);
if (posCierre != wstring::npos) {
if (posCierre != extracto.size() - 1) {
esValido = false;
}
w_extracto = w_extracto.substr(0, w_extracto.size() - 1);
}
```

```
}
}

if (w_extracto.empty()) esValido = false;
}

return {converter.to_bytes(w_extracto), esValido ? PALABRA : IGNORAR};
}

return {extracto,DESCONOCIDO};
}


ExtraccionReturn ExtraerArchivo::procesarTexto() {
istringstream streamTexto(texto);
string linea;
pair < string, Identificador > resultado;
string paginaActual;
string capituloActual;

vector <CapituloEstructura> capitulos;
vector <PalabraEstructura> palabras;

while (std::getline(streamTexto, linea)) {
++numeroDeLineas; // contabilizamos las líneas

if (linea.empty()) {
continue;
}

std::istringstream streamLinea(linea);
std::string palabra;
while (streamLinea >> palabra) {
resultado = analizarExtracto(palabra, PALABRA);

if (palabra.find("<pagina") != string::npos || palabra.find("<capitulo") !=
string::npos) {
Identificador id = palabra.find("<pagina") != string::npos ? PAGINA : CAPITULO;
resultado = analizarExtracto(linea, id);
if (resultado.second == PAGINA) {
paginaActual = resultado.first;
numeroDePaginas++;

} else if (resultado.second == CAPITULO) {
capituloActual = resultado.first;
numeroDeCapitulos++;
capitulos.push_back({ resultado.first, paginaActual });
}

} else if (palabra.find("<sub") != string::npos) {
break;
}
if (resultado.second == PALABRA) {
numeroPalabrasTotal++;
palabras.push_back({ resultado.first, paginaActual, capituloActual });
} else { // ignorar
continue;
}
}
}
Contabilizados contabilizados = {numeroDeLineas, numeroDeCapitulos,
numeroDePaginas, numeroPalabrasTotal};
return {palabras, capitulos, contabilizados};
```

```
}
```

Documento.hpp

```
#ifndef DOCUMENTO_HPP
#define DOCUMENTO_HPP

#include "Capitulos.hpp"
#include "Indice.h"
#include "ExtraerArchivo.hpp"

#include <iostream>
using namespace std;

class Documento {

private:
string texto;
string textoProcesado;
int numeroDeLineas, numeroDeCapitulos, numeroDePaginas, numeroPalabrasTotal = 0;
Indice indice;
Capitulos capitulos;
ExtraerArchivo extractor;

public:
Documento(string texto);

void procesarTexto();
// Procesa el documento para poder ser impreso por pantalla
void procesarDocumento();
void getPalabra(string nombre);
void agregarCapitulos(vector<CapituloEstructura> capitulos);
void mostrarCapitulos();
vector<string> getCapituloIndice(string nombreCapitulo);

void agregarPalabra(vector<PalabraEstructura> palabras);
vector<map<string, set<string>>> getIndice();
map<string, set<string>> buscarPalabra(string palabra);

bool eliminarPalabra(string palabra);

void setNumeroLineas(int numeroDeLineas);
int getNumeroLineas();
int getNumeroPaginas();
int getNumeroDeCapitulos();
int getNumeroPalabrasTotal();
int getNumeroPalabrasUnicas();

string getTextoProcesado();

};

#endif
```

Documento.cpp

```
#include "Documento.hpp"
#include "ExtraerArchivo.hpp"
#include <sstream>
Documento::Documento(string texto) : extractor(texto) {
this -> texto = texto;
}
```

```
void Documento::procesarTexto() {
ExtraccionReturn resultados = this -> extractor.procesarTexto();
// Distribuimos datos
this -> numeroDeLineas = resultados.contabilizados.numeroDeLineas;
this -> numeroDePaginas = resultados.contabilizados.numeroDePaginas;
this -> numeroDeCapitulos = resultados.contabilizados.numeroDeCapitulos;
this -> numeroPalabrasTotal = resultados.contabilizados.numeroDePalabrasTotal;
agregarCapitulos(resultados.capitulos);
agregarPalabra(resultados.palabras);

}

void Documento::procesarDocumento() {
istringstream streamTexto(texto);
string linea;
string textoProcesado = "";

while (std::getline(streamTexto, linea)) {
std::istringstream streamLinea(linea);
std::string palabra;
string modelo = "";
while (streamLinea >> palabra) {
if (palabra.find("<sub") != string::npos || palabra.find("<pagina") !=
string::npos) {
break;
}
if (palabra.find("<capitulo") != string::npos) {
size_t inicio = linea.find(' ') + 1;
size_t fin = linea.find('>', inicio);

if (inicio != string::npos && fin != string::npos) {
texto = linea.substr(inicio, fin - inicio);
CapituloEstructura capitulo = capitulos.buscarUnCapitulo(texto);
if (capitulo.paginaInicio != "") {
modelo = "Capitulo " + texto + " - Página " + capitulo.paginaInicio + "\n";
textoProcesado += modelo;
}
break;
}
} else {
textoProcesado += palabra + " ";
}
}
textoProcesado += "\n";
}
this-> textoProcesado = textoProcesado;
}

void Documento::agregarPalabra(vector<PalabraEstructura> palabras) {
this -> indice.insertarPalabras(palabras);
}

vector<map<string, set<string>>> Documento::getIndice() {
vector<string> llavero = this -> indice.getLlavero();
vector <map<string, set<string>>> indicePalabras; //palabra-paginas

for (string llave: llavero) {
list<tuple<string, string, string>> palabras = indice.tabla.buscar(llave);

if (!palabras.empty()) {
map<string, set<string>> wordMap;
for (const auto& trio : palabras) {
string word = get<0>(trio);
string page = get<1>(trio);
```

```
wordMap[word].insert(page);
}

indicePalabras.push_back(wordMap);
}
}

return indicePalabras;
}

map<string, set<string>> Documento::buscarPalabra(string palabra) {
return indice.busquedaParcial(palabra);
}

bool Documento::eliminarPalabra(string palabra) {
pair <bool, int> eliminado = this -> indice.eliminarPalabraIndice(palabra);
if (eliminado.first) {
this-> numeroPalabrasTotal -= eliminado.second;
}
return eliminado.second;

}

// Funciones De Capitulos

void Documento::agregarCapitulos(vector<CapituloEstructura> capitulos) {
this -> capitulos.insertarCapitulos(capitulos);
}

vector<string> Documento::getCapituloIndice(string nombreCapitulo) {
return capitulos.buscarCapituloIndice(nombreCapitulo, indice.tabla,
indice.getLlavero());
}

int Documento::getNumeroDeCapitulos() {
return this -> numeroDeCapitulos;
}

int Documento::getNumeroPaginas() {
return this -> numeroDePaginas;
}

int Documento::getNumeroLineas() {
return this -> numeroDeLineas;
}

int Documento::getNumeroPalabrasTotal() {
return this -> numeroPalabrasTotal;
}

int Documento::getNumeroPalabrasUnicas() {
return indice.getLlavero().size();
}

string Documento::getTextoProcesado() {
return this -> textoProcesado;
}
```

Capitulo.hpp

```
#ifndef CAPITULOS_HPP
#define CAPITULOS_HPP

#include "HashTable.h"
```

```
#include "ExtraerArchivo.hpp"
#include <iostream>
#include <string>

using namespace std;

class Capitulos {

private:
vector<CapituloEstructura> capitulos;
public:
Capitulos();
void insertarCapitulos(vector<CapituloEstructura> capitulos);
CapituloEstructura buscarUnCapitulo(string nombre);
vector<string> buscarCapituloIndice(string nombre, HashTable<string, string,
string> palabrasTabla, vector<string> llavero);
};

#endif
```

Capitulo.cpp

```
#include "Capitulos.hpp"
#include "ExtraerArchivo.hpp"
#include "algorithm"

Capitulos::Capitulos() {}

void Capitulos::insertarCapitulos(vector<CapituloEstructura> capitulos) {
this -> capitulos = capitulos;
}

CapituloEstructura Capitulos::buscarUnCapitulo(string nombre) {
for (CapituloEstructura capitulo : capitulos) {
if (capitulo.nombre == nombre) {
return capitulo;
}
}
return CapituloEstructura();
}

vector<string> Capitulos::buscarCapituloIndice(string nombre, HashTable<string,
string, string> palabrasTabla, vector<string> llavero) {
vector<string> palabrasEnCapitulo;

for (string llave: llavero) {
list<tuple<string, string, string>> palabras = palabrasTabla.buscar(llave);

if (!palabras.empty()) {
for (const auto& trio : palabras) {
string palabra = get<0>(trio);
string capitulo = get<2>(trio);

if (capitulo.find(nombre) != string::npos) {
if(std::find(palabrasEnCapitulo.begin(), palabrasEnCapitulo.end(), palabra) ==
palabrasEnCapitulo.end()) {
palabrasEnCapitulo.push_back(palabra);
}
}
}
}
}
sort(palabrasEnCapitulo.begin(), palabrasEnCapitulo.end());
return palabrasEnCapitulo;
```

```
}
```

GestorArchivos.hpp

```
#ifndef GESTOR_ARCHIVOS_H
#define GESTOR_ARCHIVOS_H

#include <sstream>

using namespace std;

/**
* Clase GestorDeArchivo para la manipulación con archivos.
*/
class GestorDeArchivos {

private:
stringstream buffer;
public:
std::string leerArchivo(const std::string& rutaArchivo);
};

#endif
```

GestorArchivos.cpp

```
#include "GestorDeArchivos.hpp"
#include <fstream>

std::string GestorDeArchivos::leerArchivo(const std::string& rutaArchivo) {
std::ifstream archivo(rutaArchivo);
if (!archivo) {
return "";
}

std::stringstream buffer;
buffer << archivo.rdbuf();
archivo.close();

return buffer.str();
}
```