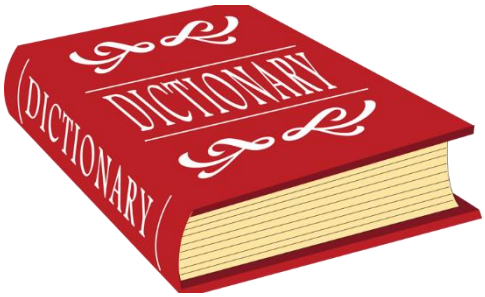
MOTOR DE BÚSQUEDA



Diccionario.h

termino cargarUnTermino(char * palabra, int idDoc, int pos)
Crea un término nuevo y llena sus campos con los datos recibidos por parámetro. Retorna el término cargado.

void mostrarUnaPalabra(termino palabra)
Imprime una palabra, su respectivo ID (número de documento) y la posición en la que se ubica.

void mostrarTodasLasPalabrasCargadas(termino* arregloDePalabras, int cantPalabras)
Se muestra todo el contenido de un arreglo con la utilización de un bucle **for** (su límite es establecido por el entero recibido por parámetro) que con cada iteración realiza un llamado a la función **mostrarUnaPalabra()**.

int contarPalabrasDelDocumento()
Lee todos los documentos disponibles y al encontrar un espacio, un punto o una coma, incrementa el contador de palabras. Retorna la cantidad total de ellas.

void guardarPalabrasDelDocumento(termino* arregloDePalabras, int* validos)
Carga el arreglo pasado por parámetro con los datos del documento. Separa el texto letra por letra y forma palabras cuando encuentra un espacio, punto o coma. Utiliza la función **cargarUnTermino()** para rellenar el arreglo.

void cargarPalabrasAlDiccionario(termino* arregloDeTerminos, int validos)
Crea un archivo de tipo diccionario. Si ya existe, lo limpia. Luego, escribe todos los datos del arreglo de términos en el mismo.

void recorrerDiccionario()
Abre el diccionario, lo lee y mediante un ciclo **while** itera la utilización de la función **mostrarUnaPalabra()** para mostrar el contenido del diccionario.

Motor.h

nodoA* crearNodoArbol(char* palabra)
Crea el nodo del árbol con la palabra recibida por parámetro.

nodoT* crearNodoOcurrencia(termino palabra)
Crea el nodo de la lista interna de los nodos del árbol.

void insertarNodoArbol(nodoA motor, termino palabra)**
Realiza una inserción por orden alfabético del término enviado por parámetro.

void insertarNodoOcurrencia(nodoT ocurrencia, termino palabra)**
Realiza una inserción ordenada por ID y posición de la lista presente dentro del árbol.

void cargarMotor(nodoA motor)**
Lee el archivo del diccionario y utiliza la función **insertarNodoArbol()** para formar el ABB.
void mostrarUnNodo(nodoA * dato)
Muestra un nodo de la lista de ocurrencias. Utiliza la función **mostrarListaDeOcurrencias()** para mostrar la cantidad de repeticiones

void mostrarMotor(nodoA* motor)
Muestra el contenido del motor (cargado por el diccionario) en modo IN-ORDER. Utiliza para esto la función **mostrarUnNodo()**.

void mostrarListaDeOcurrencias(nodoT* ocurrencias)
Muestra la lista entera de ocurrencias.

void busquedaUnica(nodoA* motor, char* palabra)
Realiza una búsqueda específica en un árbol de una palabra enviada por parámetro.

void busquedaUnica(nodoA* motor, char* palabra)
Realiza una búsqueda específica en un árbol de una palabra enviada por parámetro.

int cantPalabrasFrase(char* frase)
Cuenta la cantidad de palabras en una frase enviada por parámetro.

void mostrarUnaOcurrencia(nodoT* nodoOcurrencia)
Imprime por pantalla una única ocurrencia.

int buscarCoincidenciaID(nodoT* ocurrencias, int ID)
Recorrer la lista de ocurrencias buscando una coincidencia de ID. Si encuentra una coincidencia retorna 1, caso contrario retornará 0.

int verificarExistencias(nodoA* motor, char* frase)
Busca la frase pasada por parámetro en el motor. Utiliza la función **busquedaUnica()**.

int Levenshtein(char *s1, char * s2)
Algoritmo que revisa la distancia entre dos cadenas de caracteres.

nodoA* sugerirPalabra(nodoA * motor, char * palabra)
Utilizando la función **Levenshtein**, si la distancia entre una palabra y la existente en el motor es menor o igual a tres, sugiere una palabra que coincida con la misma.

Usuario.h

nodoP* crearNodoPalabras(char* palabra,nodoT* ocurrencias)
Crea un nodo de palabras con la palabra pasada por parámetro.

void insertarPalabraAlFinal(nodoP listaDePalabras, char* palabra, nodoT* ocurrencias)**
Inserta un nodo que posee su palabra y su respectiva lista de ocurrencias al final de la lista.

void mostrarListaDePalabras(nodoP* listaDePalabras)
Muestra el contenido de la lista de palabras.

void mostrarNodosCoincidentesDosID(nodoT* ocurrencias, int ID1, int ID2)
Busca una palabra. Si está presente en al menos uno de los documentos pasados por parámetro, la muestra.

nodoT* BuscarNodosCoincidentesUnID(nodoT* ocurrencias, int ID)
Busca una palabra mediante ID en un documento. Si coincide la retorna.

void mostrarNodosCoincidentesUnID(nodoT* ocurrencias, int ID1)
Busca una palabra enviada por parámetro en un documento. Si la encuentra la imprime con la función **mostrarUnaOcurrencia()**.

void buscarEnDosDoc(nodoA* motor, char* palabra, int ID1, int ID2)
Busca la palabra pasada por parámetro en dos documentos también pasados por parámetro. Imprime si está en uno o en el otro, en ambos o en ninguno.

void cargarPalabrasDeLaFrase(nodoA* motor,nodoPlistaPalabras,char* frase)**
Carga la lista de palabras con los elementos del motor.

int AllWordsSameDoc(nodoP* listaPalabras)
Revisa si todas las palabras de la lista de palabras se encuentran en el mismo documento. Si coinciden retorna 1.

void buscarVariosTerminosEnUnDoc(nodoA* motor,char* palabra,int ID)