



Universidad Mariano Gálvez de Guatemala

Centro Universitario de Portales

Facultad de Ingeniería en Sistemas

Programación 3

DOCUMENTACION PROYECTO FINAL

Proyecto: Traductor c++

Integrantes de grupo: Pablo Jose Ramirez Rac 9989-22-4989

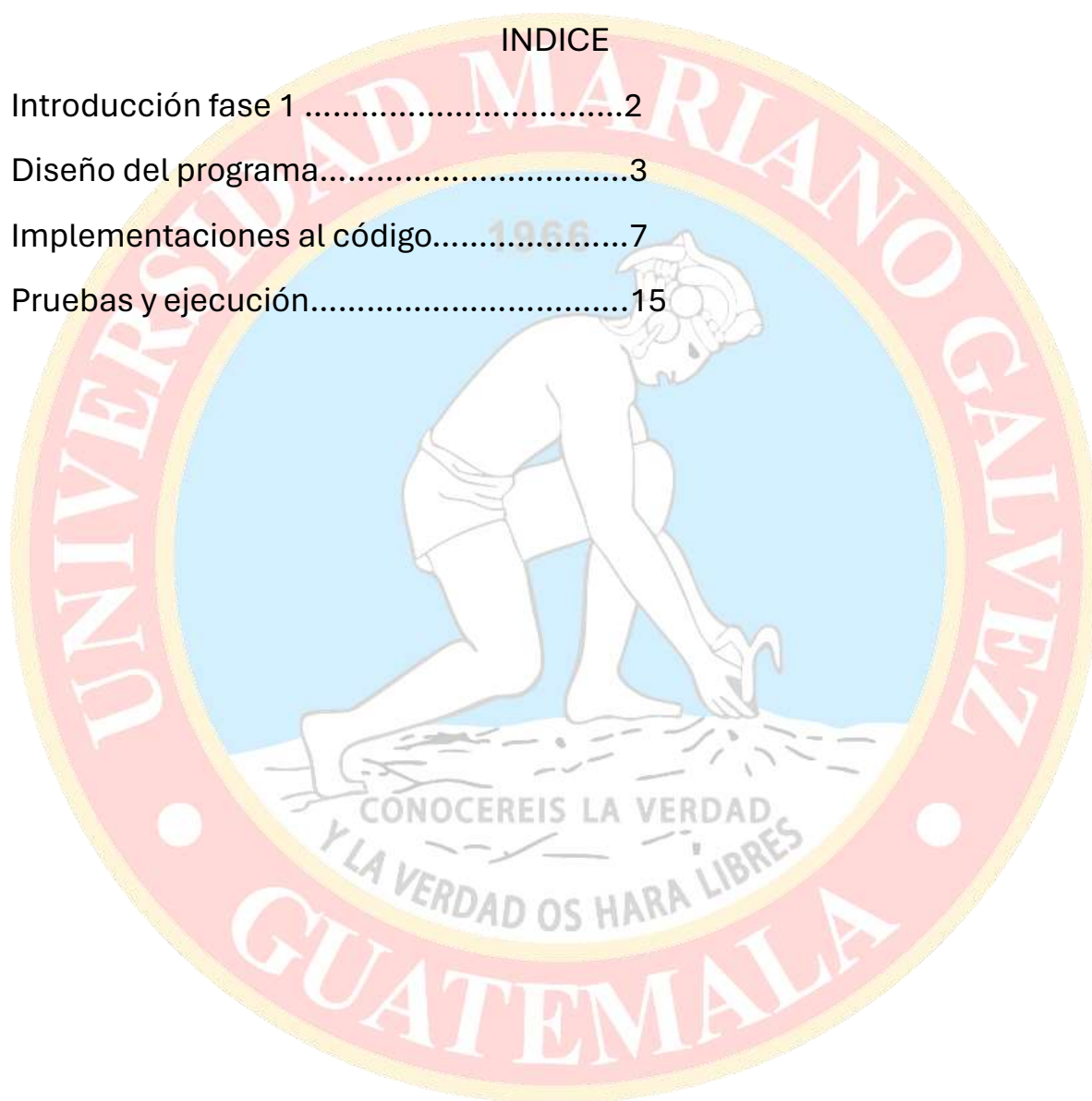
Elvin Eduardo Muralles Melgar 9989-22-5053

Jeffry Alexander Navarro Muralles 9989-22-5310

Fecha: 25 de mayo de 2024

## INDICE

Introducción fase 1 .....	2
Diseño del programa.....	3
Implementaciones al código.....	7
Pruebas y ejecución.....	15



## INTRODUCCION FASE 1

En la fase uno del proyecto, hemos creado una aplicación en C++ que cumple con los requisitos especificados para desarrollar un traductor de palabras del español a italiano, francés, alemán e inglés. Este traductor es capaz de cargar la información desde una base de datos que contiene registros estructurados de palabras en español y sus respectivas traducciones a los idiomas mencionados.

La aplicación permite realizar las siguientes acciones:

Lectura de un archivo el cual tendrá las traducciones correspondientes a cada idioma asignado, agregar nuevos nodos, eliminar nodos y

Reproducir audio (Esto mejora la experiencia del usuario al proporcionar una forma adicional de comprensión y pronunciación de las palabras).

Siguiendo con los otros parámetros que se nos fue indicado en cada fase, logramos hacer la realización de nuestro programa con el código que será presentado en clase.

## DISEÑO DEL PROGRAMA

Como primer paso en la creación de nuestro programa fue la implementación de los árboles, en este caso utilizamos el método de los árboles AVL.

```
Nodo: struct
Usuario: struct
altura(Nodo* nodo) : int
buscarPalabra(Nodo* nodo, const string& palabra_es) : Nodo*
cargarPalabrasDesdeArchivo(Nodo*& raiz, const string& palabras) : void
cargarUsuarios() : vector
crearUsuario(vector<Usuario>& usuarios) : void
eliminarNodo(Nodo* raiz, string palabra_es) : Nodo*
eliminarUsuario(vector<Usuario>& usuarios) : void
encontrarNodoMinimo(Nodo* nodo) : Nodo*
encriptarPalabra(const string& palabra) : string
guardarPalabraEncriptada(const string& palabra, const string& palabraEncriptada) : void
guardarUsuarios(const vector<Usuario>& usuarios) : void
ingresarUsuario(vector<Usuario>& usuarios) : bool
insertarNodo(Nodo* nodo, string es, string it, string fr, string de, string en, const string& archivo) : Nodo*
main() : int
maximo(int a, int b) : int
mostrarMenu(Nodo* raiz, vector<Usuario>& usuarios) : void
mostrarMenuUsuarios(vector<Usuario>& usuarios) : void
mostrarTraducciones(Nodo* nodo) : void
obtenerBalance(Nodo* nodo) : int
```

Al ejecutar nuestro programa nos muestra un menú con diferentes opciones las cuales son las siguientes:

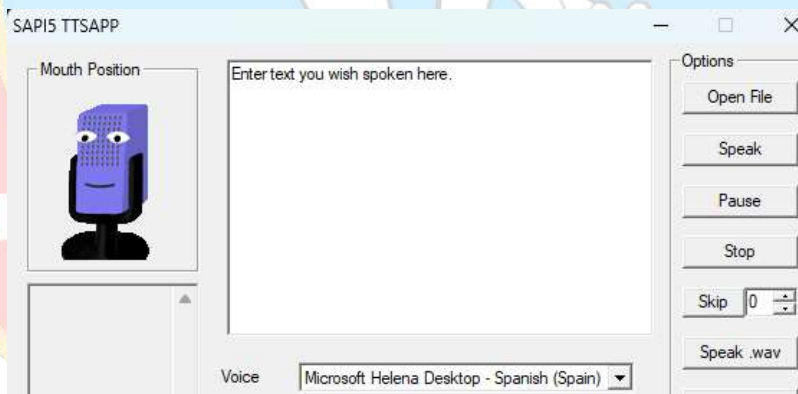
```
C:\archivos+\traductor\Trad x + v
1. Buscar palabra
2. Agregar nueva palabra
3. Eliminar palabra
4. Salir
Ingresa una opcion:
```

Comenzamos con la primera opción del menú (1. Buscar palabra), esta opción nos permite ingresar una palabra la cual nos mostrará su traducción en los diferentes tipos de idiomas y abrirá el archivo ya creado para la búsqueda de la palabra correspondiente, por ejemplo:

```
Ingrese la palabra a buscar: hola
```

```
Ingrese la palabra a buscar: hola
Palabra encontrada: hola
Traducciones:
Aleman: Ciao
Frances: Hola
Ingles: Bonjour
Italiano: hola
```

“cada palabra tiene su respectiva traducción por medio de una voz artificial” haciendo énfasis esto es gracias a un programa externo “eSpeak” el cual fue implementado dentro del programa.





La opción dos (2. Agregar nueva palabra) esta opción nos permitirá la implementación de una nueva palabra que será guardada en el archivo correspondiente, en este caso debemos agregar la palabra en español con sus diferentes traducciones, por ejemplo:

```
int main() {  
    Nodo* raiz = nullptr;  
    cargarPalabrasDesdeArchivo(raiz, "palabras.txt");  
}
```

```
Ingrese la palabra para agregar: taza  
Ingrese la traducción en italiano: tacsá  
Ingrese la traducción en frances: tauza  
Ingrese la traducción en aleman: tose  
Ingrese la traducción en ingles: taza
```

La palabra será agregada y la podemos visualizar con la opción (1. Buscar palabra)

```
Ingrese la palabra a buscar: taza  
Palabra encontrada: taza  
Traducciones:  
Aleman: tose  
Frances: tauza  
Ingles: taza  
Italiano: tacsá
```

Por último, pero no menos importante, tenemos la opción (3. Eliminar palabra), esta opción nos permite eliminar una palabra no deseada dentro del archivo y eliminará registro de sus traducciones, por ejemplo:

```
Ingrese la palabra en español a eliminar: taza
```

## IMPLEMENTACIONES AL CÓDIGO DEL PROGRAMA

### FASE 2

En este apartado mostraremos las condiciones que nos indica la fase número dos del proyecto la cual tiene los siguientes criterios:

Agregar una estructura de árbol AVL para la búsqueda de palabras.

Para esto necesitamos guardar un historial de las palabras buscadas, en este caso crearemos el historial con las palabras más buscadas por el usuario. También se nos pide la encriptación de palabras buscadas por el usuario.

Para el proceso de Encriptación se le sugiere el siguiente:

Palabra clave:

Umg → U representa a todas las vocales como se explica a continuación  
m representa a todas las letras minúsculas del alfabeto menos las vocales

g representa a todas las letras mayúsculas del alfabeto menos las vocales

Leer la cadena de texto y reemplazar las vocales y letras.

b	m1	B	g1
c	m2	C	g2
d	m3	D	g3
f	m4	F	g4
g	m5	G	g5
h	m6	H	g6
j	m7	J	g7
k	m8	K	g8
l	m9	L	g9
m	m10	M	g10
n	m11	N	g11
ñ	m12	Ñ	g12
p	m13	P	g13
q	m14	Q	g14
r	m15	R	g15
s	m16	S	g16
t	m17	T	g17
v	m18	V	g18
w	m19	W	g19
x	m20	X	g20
y	m21	Y	g21
z	m22	Z	g22

a	U1
e	U2
i	U3
o	U4
u	U5

Para este caso creamos la tabla correspondiente a los criterios dentro del código junto con su función para la encriptación de las palabras buscadas por el usuario.

```
string encriptarPalabra(const string& palabra) {
    map<char, string> tablaEncriptacion = {{'a', "U1"}, {'e', "U2"}, {'i', "U3"}, {'o', "U4"}, {'u', "U5"},
                                           {'b', "m1"}, {'c', "m2"}, {'d', "m3"}, {'f', "m4"}, {'g', "m5"},
                                           {'h', "m6"}, {'j', "m7"}, {'k', "m8"}, {'l', "m9"}, {'m', "m10"},
                                           {'n', "m11"}, {'p', "m12"}, {'q', "m13"}, {'r', "m14"}, {'s', "m15"},
                                           {'t', "m16"}, {'v', "m17"}, {'w', "m18"}, {'x', "m19"}, {'y', "m20"},
                                           {'z', "m21"}, {'A', "G1"}, {'B', "G2"}, {'C', "G3"}, {'D', "G4"},
                                           {'E', "G5"}, {'F', "G6"}, {'G', "G7"}, {'H', "G8"}, {'I', "G9"},
                                           {'J', "G10"}, {'K', "G11"}, {'L', "G12"}, {'M', "G13"}, {'N', "G14"},
                                           {'O', "G15"}, {'P', "G16"}, {'Q', "G17"}, {'R', "G18"}, {'S', "G19"},
                                           {'T', "G20"}, {'U', "G21"}, {'V', "G22"}, {'W', "G23"}, {'X', "G24"},
                                           {'Y', "G25"}, {'Z', "G26"}, {' ', "k00"}, {'.', "k01"}, {'.', "k02"}};

    string palabraEncriptada;
```

Para esta implementación dentro del código tenemos la función de visualizar palabras encriptadas, cabe recalcar que estas palabras encriptadas son las que nosotros registraremos dentro de la ejecución, y este proceso lo lograremos realizar gracias a switch el cual agregará la palabra para luego ser mostrada.

```
switch (opcion) {
    case 1: {
        cout << "Ingrese la palabra a buscar: ";
        cin >> palabra_es;
        Nodo* encontrado = buscarPalabra(raiz, palabra_es);
        if (encontrado != nullptr) {
            cout << "Palabra encontrada: " << encontrado->palabra_es << endl;
            mostrarTraducciones(encontrado);
            historial.push_back(palabra_es);
            contadorPalabras[palabra_es]++;
            string palabra_encriptada = encriptarPalabra(palabra_es);
            guardarPalabraEncriptada(palabra_es, palabra_encriptada);
        } else {
            cout << "Palabra no encontrada." << endl;
        }
        break;
    }
```

Ahora, con el diseño visual o ejecutable del programa se nos muestra la siguiente opción en el menú:

(5. Visualizar palabras encriptadas)

**5. Visualizar palabras encriptadas**



Al momento de seleccionar la opción nos permite ingresar la palabra la cual agregamos recientemente con su respectiva encriptación, ejemplo:

```
Palabras encriptadas:
Original: hola, Encriptada: m6U4m9U1
Original: hola, Encriptada: m6U4m9U1
Original: taza, Encriptada: m16U1m21U1
```

NOTA: las palabras encriptadas serán dirigidas a un nuevo archivo el cual contendrá las palabras recientemente agregadas para que nos la vuelva a mostrar.

```
visualizarPalabrasEncriptadas("historial_encriptado.txt");
```

Siguiendo con los criterios de la fase numero dos, se nos pide asignar al programa un historial de búsqueda dentro del menú, esto nos permitirá mostrar las palabras más buscadas por el usuario y lo logramos utilizando un ciclo el cual nos muestra la recién palabra agregada dentro de la opción.

```
cout << "\nHistorial de palabras buscadas:\n";
for (const auto& palabra : historial) {
    cout << palabra << endl;
}
```

En el menú se nos muestra la siguiente opción (4. Ver historial y sugerencias)

4. Ver historial y sugerencias

```
Historial de palabras buscadas:
taza
Palabras sugeridas (mbs buscadas):
taza (1 veces)
```

En este caso hemos trabajado con la palabra taza como ejemplo y nos muestra esta palabra como ejemplo el cual la hemos utilizado una vez.

### FASE 3.

En esta fase se busca darle una mejor funcionalidad a la aplicación, permitiendo garantizar la confidencialidad de los perfiles creado, esto mediante la creación de usuarios en la aplicación. Se realizan las siguientes acciones:

- permitir guardar la información de los usuarios salvaguardando esta con el proceso de encriptación de la fase 2.

Para este proceso utilizamos nuevamente un archivo de texto donde quedarán registrados los usuarios ingresados.

```
void guardarUsuarios(const vector<Usuario>& usuarios) {
    ofstream archivo("usuarios.txt");
    for (const auto& usuario : usuarios) {
        archivo << usuario.nombre << " " << usuario.contrasena << endl;
    }
    archivo.close();
}
```

### PROCESO DE CREACIÓN DEL USUARIO

Para crear el usuario, se nos abre un menú principal antes de mostrar las implementaciones 1 y 2 donde se nos pide ingresar un usuario si ya estuviese creado o bien crear uno para el inicio de sesión. Como se muestra en la siguiente imagen.

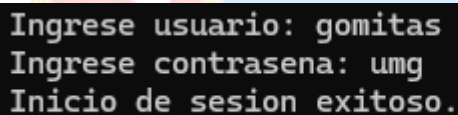
```
-----
1. Iniciar sesion
2. Registrar usuario
3. Eliminar usuario
4. Visualizar usuarios registrados
5. Salir
-----
```

```
Ingresa una opcion: |
```

En la opción 1. Tenemos la posibilidad de ingresar con un usuario y contraseña si es que este ya esté grabado y guardado en el archivo para poder ingresar. Esto lo logramos con la siguiente función

```
bool iniciarSesion(const string& usuario, const string& contrasena) {  
    ifstream archivo("usuarios.txt");  
    string u, c;  
    while (archivo >> u >> c) {  
        if (u == usuario && c == contrasena) {  
            return true;  
        }  
    }  
    return false;  
}
```

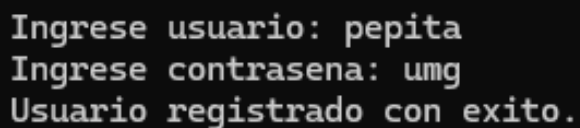
Ejemplo:



```
Ingrese usuario: gomitas  
Ingrese contrasena: umg  
Inicio de sesion exitoso.  
-----
```

Lo siguiente que se muestra es la función para la creación y registro de los usuarios y contraseñas.

En la opción 2. Tenemos la posibilidad de crear un usuario nuevo con las credenciales de nuestra preferencia, como se muestra a continuación la creación de un nuevo usuario.

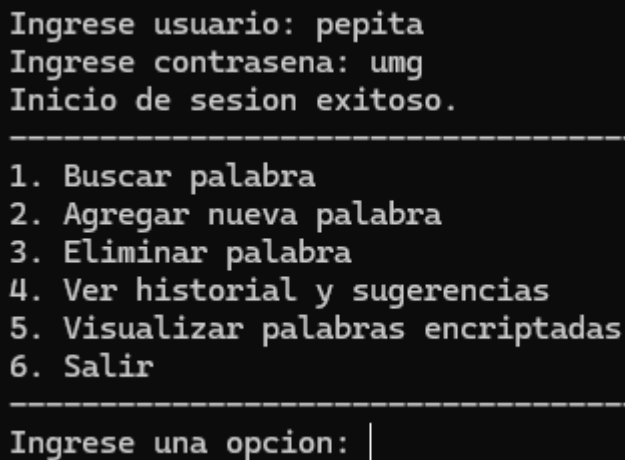


```
Ingrese usuario: pepita  
Ingrese contrasena: umg  
Usuario registrado con exito.  
-----
```

El usuario será registrado exitosamente dentro del archivo y nos permite avanzar a la opción de ingresar gracias a la siguiente función.

```
void registrarUsuario(const string& usuario, const string& contrasena) {
    ofstream archivo("usuarios.txt", ios::app);
    archivo << usuario << " " << contrasena << endl;
    archivo.close();
}
```

Al momento de colocar nuestro usuario nos permite seguir navegando por las otras opciones correspondientes al programa.



```
Ingrese usuario: pepita
Ingrese contrasena: umg
Inicio de sesion exitoso.

-----
1. Buscar palabra
2. Agregar nueva palabra
3. Eliminar palabra
4. Ver historial y sugerencias
5. Visualizar palabras encriptadas
6. Salir
-----
Ingrese una opcion: |
```

luego tenemos otra opción 3. Corresponde a la posibilidad de eliminar un usuario ya registrado, esto lo aremos con la siguiente función:

```
case 3:
    cout << "Ingrese el usuario a eliminar: ";
    cin >> usuario;
    eliminarUsuario(usuario);
    cout << "Usuario eliminado con exito." << endl;
    break;
```

Utilizamos la función:

```
void eliminarUsuario(const string& usuario) {
    ifstream archivo("usuarios.txt");
    ofstream archivo_temp("temp.txt");
    string u, c;
    while (archivo >> u >> c) {
        if (u != usuario) {
            archivo_temp << u << " " << c << endl;
        }
    }
    archivo.close();
    archivo_temp.close();
    remove("usuarios.txt");
    rename("temp.txt", "usuarios.txt");
}
```

Y si dando el ejemplo, procedemos a eliminar el usuario anterior.

```
Ingrese el usuario a eliminar: pepita
Usuario eliminado con exito.
```

```
-----
1. Iniciar sesion
```

```
Ingrese usuario:
pepita
Ingrese contrasena: umg
Usuario o contrasena incorrectos.
```

Luego tenemos la función de la opción 4 el cual nos mostrará los usuarios creados:

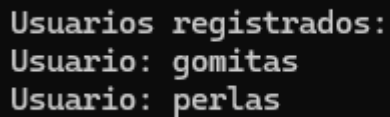
```
case 4:
    visualizarUsuarios();
    break;
```



Con la siguiente función, nos permite abrir el archivo donde están registrados los usuarios para posteriormente mostrárnolos

```
void visualizarUsuarios() {  
    ifstream archivo("usuarios.txt");  
    string usuario, contrasena;  
    cout << "Usuarios registrados:" << endl;  
    while (archivo >> usuario >> contrasena) {  
        cout << "Usuario: " << usuario << endl;  
    }  
    archivo.close();  
}
```

Ejemplo:



```
Usuarios registrados:  
Usuario: gomitas  
Usuario: perlas  
-----
```

Cuando intentamos INGRESAR CON OTRO USUARIO nos muestra que no podemos ingresar por lo cual no nos permitirá seguir navegando

GRACIAS.

## PRUEBAS DE EJECUCION Y CÓDIGO

```
Ingrese usuario: hola
Ingrese contrasena: umg
Inicio de sesion exitoso.
-----
1. Buscar palabra
2. Agregar nueva palabra
3. Eliminar palabra
4. Ver historial y sugerencias
5. Visualizar palabras encriptadas
6. Salir
-----
Ingrese una opcion: |
```

```
Ingrese la palabra a buscar: hola
Palabra encontrada: hola
Traducciones:
Aleman: Ciao
Frances: Hola
Ingles: Bonjour
Italiano: hola
1. Buscar palabra
2. Agregar nueva palabra
3. Eliminar palabra
4. Ver historial y sugerencias
5. Visualizar palabras encriptadas
6. Menu de usuarios
7. Salir
Ingrese una opcion: |
```

```
Ingrese la palabra para agregar: carro
Ingrese la traducci3n en italiano: caru
Ingrese la traducci3n en frances: caroe
Ingrese la traducci3n en aleman: cari
Ingrese la traducci3n en ingles: carr
1. Buscar palabra
```

```

Historial de palabras buscadas:
hola
taza
Palabras sugeridas (mfs buscadas):
hola (1 veces)
taza (1 veces)

```

```

Palabras encriptadas:
Original: hola, Encriptada: m6U4m9U1
Original: hola, Encriptada: m6U4m9U1
Original: taza, Encriptada: m16U1m21U1
Original: hola, Encriptada: m6U4m9U1
Original: taza, Encriptada: m16U1m21U1
1. Buscar palabra

```

```

Saliendo del programa...

```

```

-----
Process exited after 143.9 seconds with return value 0
Presione una tecla para continuar . . . |

```

```

#include <iostream>
#include <fstream>
#include <string>
#include <map>
#include <vector>
#include <algorithm>

using namespace std;

struct Nodo {
    string palabra_es;
    map<string, string> traducciones;
    Nodo* izquierdo;
    Nodo* derecho;
    int altura;

```

```

Nodo(string es, string it, string fr, string de, string en) {

    palabra_es = es;

    traducciones["Italiano"] = it;

    traducciones["Frances"] = fr;

    traducciones["Aleman"] = de;

    traducciones["Ingles"] = en;

    izquierdo = nullptr;

    derecho = nullptr;

    altura = 1;

    1966 }

    ~Nodo() {

        delete izquierdo;

        delete derecho;

    }

    };

    vector<string> historial_encryptado;

    map<string, int> contadorPalabras;

    struct Usuario {

        string nombre;

        string contrasena;

        // Otros datos del usuario, como permisos, historial, etc.

    };

    int altura(Nodo* nodo) {

        if (nodo == nullptr)

            return 0;

        return nodo->altura;

    }

    int maximo(int a, int b) {

        return (a > b) ? a : b;
    
```

```
}
```

```
int obtenerBalance(Nodo* nodo) {
    if (nodo == nullptr)
        return 0;
    return altura(nodo->izquierdo) - altura(nodo->derecho);
}
```

```
Nodo* rotacionDerecha(Nodo* y) {
    Nodo* x = y->izquierdo;
    Nodo* T2 = x->derecho;

    x->derecho = y;
    y->izquierdo = T2;

    y->altura = maximo(altura(y->izquierdo), altura(y->derecho)) + 1;
    x->altura = maximo(altura(x->izquierdo), altura(x->derecho)) + 1;

    return x;
}
```

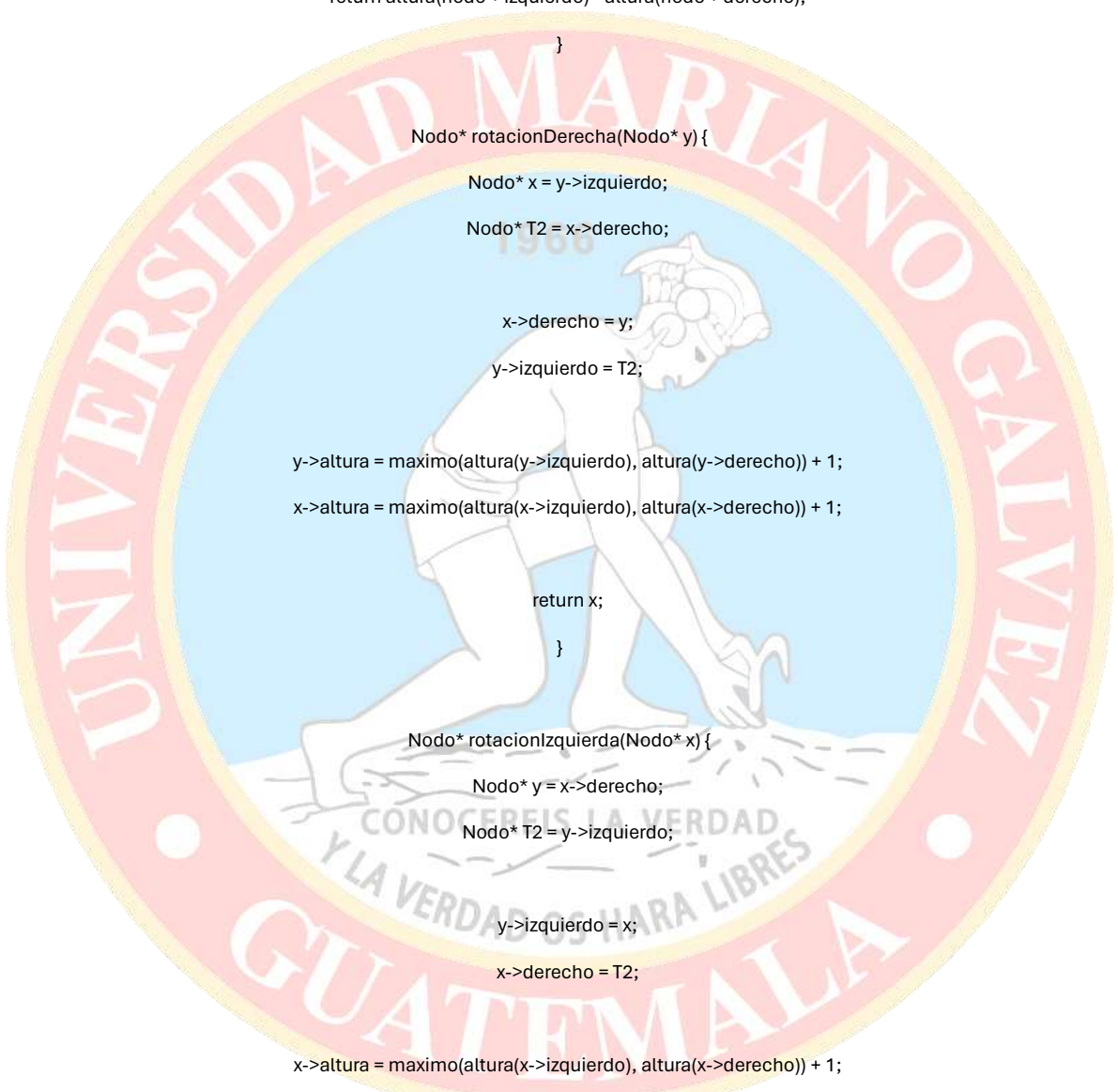
```
Nodo* rotacionIzquierda(Nodo* x) {
    Nodo* y = x->derecho;
    Nodo* T2 = y->izquierdo;

    y->izquierdo = x;
    x->derecho = T2;

    x->altura = maximo(altura(x->izquierdo), altura(x->derecho)) + 1;
    y->altura = maximo(altura(y->izquierdo), altura(y->derecho)) + 1;
```

```
    return y;
}
```

```
Nodo* insertarNodo(Nodo* nodo, string es, string it, string de, string en, const string& archivo) {
```





```

        if (nodo == nullptr) {

            nodo = new Nodo(es, it, fr, de, en);

            ofstream archivo_salida(archivo, ios::app);

            archivo_salida << es << " " << it << " " << fr << " " << de << " " << en << endl;

            archivo_salida.close();

            return nodo;

        }

        if (es < nodo->palabra_es)

            nodo->izquierdo = insertarNodo(nodo->izquierdo, es, it, fr, de, en, archivo);

        else if (es > nodo->palabra_es)

            nodo->derecho = insertarNodo(nodo->derecho, es, it, fr, de, en, archivo);

        else

            return nodo;

        nodo->altura = 1 + maximo(altura(nodo->izquierdo), altura(nodo->derecho));

        int balance = obtenerBalance(nodo);

        if (balance > 1 && es < nodo->izquierdo->palabra_es)

            return rotacionDerecha(nodo);

        if (balance < -1 && es > nodo->derecho->palabra_es)

            return rotacionIzquierda(nodo);

        if (balance > 1 && es > nodo->izquierdo->palabra_es) {

            nodo->izquierdo = rotacionIzquierda(nodo->izquierdo);

            return rotacionDerecha(nodo);

        }

        if (balance < -1 && es < nodo->derecho->palabra_es) {

            nodo->derecho = rotacionDerecha(nodo->derecho);

            return rotacionIzquierda(nodo);

        }
    }

```

```

return nodo;

}

Nodo* encontrarNodoMinimo(Nodo* nodo){

    Nodo* actual = nodo;

    while (actual->izquierdo != nullptr)

        actual = actual->izquierdo;

    return actual;

}

Nodo* eliminarNodo(Nodo* raiz, string palabra_es){

    if (raiz == nullptr)

        return raiz;

    if (palabra_es < raiz->palabra_es)

        raiz->izquierdo = eliminarNodo(raiz->izquierdo, palabra_es);

    else if (palabra_es > raiz->palabra_es)

        raiz->derecho = eliminarNodo(raiz->derecho, palabra_es);

    else {

        if (raiz->izquierdo == nullptr || raiz->derecho == nullptr) {

            Nodo* temp = raiz->izquierdo ? raiz->izquierdo : raiz->derecho;

            if (temp == nullptr) {

                temp = raiz;

                raiz = nullptr;

            } else {

                *raiz = *temp;

                delete temp;

            } else {

                Nodo* temp = encontrarNodoMinimo(raiz->derecho);

                raiz->palabra_es = temp->palabra_es;

                raiz->derecho = eliminarNodo(raiz->derecho, temp->palabra_es);

            }

        }

    }

}

```

```
if (raiz == nullptr)
```

```
    return raiz;
```

```
    raiz->altura = 1 + maximo(altura(raiz->izquierdo), altura(raiz->derecho));
```

```
    int balance = obtenerBalance(raiz);
```

```
    if (balance > 1 && obtenerBalance(raiz->izquierdo) >= 0)
```

```
        return rotacionDerecha(raiz);
```

```
    if (balance > 1 && obtenerBalance(raiz->izquierdo) < 0) {
```

```
        raiz->izquierdo = rotacionIzquierda(raiz->izquierdo);
```

```
        return rotacionDerecha(raiz);
```

```
    }
```

```
    if (balance < -1 && obtenerBalance(raiz->derecho) <= 0)
```

```
        return rotacionIzquierda(raiz);
```

```
    if (balance < -1 && obtenerBalance(raiz->derecho) > 0) {
```

```
        raiz->derecho = rotacionDerecha(raiz->derecho);
```

```
        return rotacionIzquierda(raiz);
```

```
    }
```

```
    return raiz;
```

```
}
```

```
Nodo* buscarPalabra(Nodo* nodo, const string& palabra_es) {
```

```
    if (nodo == nullptr || nodo->palabra_es == palabra_es)
```

```
        return nodo;
```

```
    if (palabra_es < nodo->palabra_es)
```

```
        return buscarPalabra(nodo->izquierdo, palabra_es);
```

```
    else
```

```
        return buscarPalabra(nodo->derecho, palabra_es);
```

```
}
```

```

void cargarPalabrasDesdeArchivo(Nodo*& raiz, const string& palabras) {
    ifstream archivo(palabras);

    string es, it, fr, de, en;

    while (archivo >> es >> it >> fr >> de >> en) {
        raiz = insertarNodo(raiz, es, it, fr, de, en, palabras);
    }
}

void reproducirVoz(const string& palabra) {
    string ruta_speak = "C:\\eSpeak\\command_line\\espeak.exe";
    string comando = ruta_speak + " \"" + palabra + "\"";
    system(comando.c_str());
}

void mostrarTraducciones(Nodo* nodo) {
    cout << "Traducciones:" << endl;
    for (const auto& traduccion : nodo->traducciones) {
        cout << traduccion.first << ": " << traduccion.second << endl;
        reproducirVoz(traduccion.second);
    }
}

string encriptarPalabra(const string& palabra) {
    map<char, string> tablaEncriptacion = {{'a', "U1"}, {'e', "U2"}, {'i', "U3"}, {'o', "U4"}, {'u', "U5"},
        {'b', "m1"}, {'c', "m2"}, {'d', "m3"}, {'f', "m4"}, {'g', "m5"},
        {'h', "m6"}, {'j', "m7"}, {'k', "m8"}, {'l', "m9"}, {'m', "m10"},
        {'n', "m11"}, {'p', "m12"}, {'q', "m13"}, {'r', "m14"}, {'s', "m15"},
        {'t', "m16"}, {'v', "m17"}, {'w', "m18"}, {'x', "m19"}, {'y', "m20"},
        {'z', "m21"}, {'A', "G1"}, {'B', "G2"}, {'C', "G3"}, {'D', "G4"},
        {'E', "G5"}, {'F', "G6"}, {'G', "G7"}, {'H', "G8"}, {'I', "G9"},
        {'J', "G10"}, {'K', "G11"}, {'L', "G12"}, {'M', "G13"}, {'N', "G14"},
        {'O', "G15"}, {'P', "G16"}, {'Q', "G17"}, {'R', "G18"}, {'S', "G19"},
        {'T', "G20"}, {'U', "G21"}, {'V', "G22"}, {'W', "G23"}, {'X', "G24"},
        {'Y', "G25"}, {'Z', "G26"}, {' ', "k00"}, {'.', "k01"}, {',', "k02"};
}

```



```

        string palabraEncriptada;

        for (char letra : palabra) {

            palabraEncriptada += tablaEncriptacion[letra];

        }

        return palabraEncriptada;

    }

void guardarPalabraEncriptada(const string& palabra, const string& palabraEncriptada) {

    ofstream archivoEncriptado("historial_encriptado.txt", ios::app);

    archivoEncriptado << palabra << " " << palabraEncriptada << endl;

    archivoEncriptado.close();

}

void visualizarPalabrasEncriptadas(const string& archivo) {

    ifstream archivoEntrada(archivo);

    string palabra, palabraEncriptada;

    cout << "\nPalabras encriptadas:\n";

    while (archivoEntrada >> palabra >> palabraEncriptada) {

        cout << "Original: " << palabra << ", Encriptada: " << palabraEncriptada << endl;

    }

}

void mostrarMenuUsuarios(vector<Usuario>& usuarios);

void mostrarMenu(Nodo* raiz, vector<Usuario>& usuarios);

// Función para crear un nuevo usuario

void crearUsuario(vector<Usuario>& usuarios) {

    string nombre, contrasena;

    cout << "Ingrese el nombre de usuario: ";

    cin >> nombre;

    cout << "Ingrese la contraseña: ";

    cin >> contrasena;

    usuarios.push_back({nombre, contrasena});

    cout << "Usuario creado exitosamente.\n";

```



```

    }

    // Función para eliminar un usuario
    void eliminarUsuario(vector<Usuario>& usuarios) {
        string nombre;
        cout << "Ingrese el nombre de usuario a eliminar: ";
        cin >> nombre;
        auto it = find_if(usuarios.begin(), usuarios.end(), [&nombre](const Usuario& usuario) {
            return usuario.nombre == nombre;
        });
        if (it != usuarios.end()) {
            usuarios.erase(it);
            cout << "Usuario eliminado exitosamente.\n";
        } else {
            cout << "Usuario no encontrado.\n";
        }
    }

    // Función para iniciar sesión
    bool ingresarUsuario(vector<Usuario>& usuarios) {
        string nombre, contrasena;
        cout << "Ingrese el nombre de usuario: ";
        cin >> nombre;
        cout << "Ingrese la contraseña: ";
        cin >> contrasena;
        for (const auto& usuario : usuarios) {
            if (usuario.nombre == nombre && usuario.contrasena == contrasena) {
                cout << "Inicio de sesión exitoso.\n";
                return true;
            }
        }

        cout << "Nombre de usuario o contraseña incorrectos.\n";
        return false;
    }

```

```

void guardarUsuarios(const vector<Usuario>& usuarios) {

    ofstream archivo("usuarios.txt");

    for (const auto& usuario : usuarios) {

archivo << usuario.nombre << " " << usuario.contrasena << endl;

    }

    archivo.close();

}

vector<Usuario> cargarUsuarios() {

    vector<Usuario> usuarios;

    ifstream archivo("usuarios.txt");

    string nombre, contrasena;

    while (archivo >> nombre >> contrasena) {

usuarios.push_back({nombre, contrasena});

    }

    archivo.close();

    return usuarios;

}

// Función para mostrar el menú de usuarios
void mostrarMenuUsuarios(vector<Usuario>& usuarios) {

    int opcion = 0;

    do {

        cout << "\nMenu de Usuarios\n";
        cout << "1. Crear nuevo usuario\n";
        cout << "2. Ingresar usuario\n";
        cout << "3. Eliminar usuario\n";
        cout << "4. Visualizar usuarios registrados\n";
        cout << "5. Salir\n";

        cout << "Ingrese una opcion: ";

        cin >> opcion;

        switch (opcion) {

            case 1:

                crearUsuario(usuarios);

```

```

        break;

        case 2:

            if (ingresarUsuario(usuarios)){

                return; // Salir del menú de usuarios y proceder al menú principal

            }

            break;

        case 3:

            eliminarUsuario(usuarios);

            break;

        case 4:

            cout << "\nUsuarios Registrados:\n";

            for (const auto& usuario : usuarios) {

                cout << "Nombre: " << usuario.nombre << ", Contraseña: " << usuario.contrasena << endl;

            }

            break;

        case 5:

            cout << "Saliendo...\n";

            break;

        default:

            cout << "Opción no válida. Intente de nuevo.\n";

            }

        } while (opcion != 5);

        guardarUsuarios(usuarios); // Guardar usuarios al salir del menú

    }

    void mostrarMenu(Nodo* raiz, vector<Usuario>& usuarios) {

        int opcion = 0;

        string palabra_es, palabra_it, palabra_fr, palabra_de, palabra_en;

        vector<string> historial;

        do {

            cout << "1. Buscar palabra\n";

            cout << "2. Agregar nueva palabra\n";

            cout << "3. Eliminar palabra\n";

```

```

cout << "4. Ver historial y sugerencias\n";

cout << "5. Visualizar palabras encriptadas\n";

cout << "6. Menu de usuarios\n";

cout << "7. Salir\n";

cout << "Ingrese una opcion: ";

cin >> opcion;

system("cls");

switch (opcion) {

    case 1: {

        cout << "Ingrese la palabra a buscar: ";

        cin >> palabra_es;

        Nodo* encontrado = buscarPalabra(raiz, palabra_es);

        if (encontrado != nullptr) {

            cout << "Palabra encontrada: " << encontrado->palabra_es << endl;

            mostrarTraducciones(encontrado);

            historial.push_back(palabra_es);

            contadorPalabras[palabra_es]++;

            string palabra_encriptada = encriptarPalabra(palabra_es);

            guardarPalabraEncriptada(palabra_es, palabra_encriptada);

        } else {

            cout << "Palabra no encontrada." << endl;

        }

        break;

    }

    case 2: {

        cout << "Ingrese la palabra para agregar: ";

        cin >> palabra_es;

        cout << "Ingrese la traducción en italiano: ";

        cin >> palabra_it;

        cout << "Ingrese la traducción en frances: ";

        cin >> palabra_fr;

        cout << "Ingrese la traducción en aleman: ";

        cin >> palabra_de;

        cout << "Ingrese la traducción en ingles: ";

```



```

        cin >> palabra_en;

        raiz = insertarNodo(raiz, palabra_es, palabra_it, palabra_fr, palabra_de, palabra_en, "palabras.txt");

        historial_encryptado.push_back(encryptarPalabra(palabra_es));

        break;

    }

    case 3: {

        cout << "Ingrese la palabra en español a eliminar: ";

        cin >> palabra_es;

        raiz = eliminarNodo(raiz, palabra_es);

        break;

    }

    case 4: {

        cout << "\nHistorial de palabras buscadas:\n";

        for (const auto& palabra : historial) {

            cout << palabra << endl;

        }

        cout << "Palabras sugeridas (más buscadas):\n";

        vector<pair<string, int>> sugerencias(contadorPalabras.begin(), contadorPalabras.end());

        sort(sugerencias.begin(), sugerencias.end(), [](const pair<string, int>& a, const pair<string, int>& b) {

            return a.second > b.second;

        });

        int top = min(3, static_cast<int>(sugerencias.size()));

        for (int i = 0; i < top; ++i) {

            cout << sugerencias[i].first << " (" << sugerencias[i].second << " veces)" << endl;

        }

        break;

    }

    case 5: {

        visualizarPalabrasEncriptadas("historial_encryptado.txt");

        break;

    }

    case 6: {

        mostrarMenuUsuarios(usuarios);

        break;

    }

```



```

        case 7: {
            cout << "Saliendo del programa..." << endl;

            break;

        }

        default:

            cout << "Opción no válida. Intente de nuevo." << endl;

    }

    } while (opcion != 7);

    ofstream archivo_encriptado("historial_encriptado.txt");

    for (const string& palabra : historial_encriptado) {

        archivo_encriptado << palabra << endl;

    }

    archivo_encriptado.close();

}

int main() {

    Nodo* raiz = nullptr;

    cargarPalabrasDesdeArchivo(raiz, "palabras.txt");

    vector<Usuario> usuarios = cargarUsuarios(); // Cargar usuarios al iniciar el programa

    mostrarMenuUsuarios(usuarios);

    mostrarMenu(raiz, usuarios);

    delete raiz;

    return 0;

}

```

