



ESTRUCTURA DE DATOS

Proyecto: Árbol Genealógico

Docente:

Mgt. Harry Yeison Gonzales Condori

Integrantes:

- Arenas Chullo Juan Guillermo
- Farfan Ccoto Dayana Stephany
- Palomino Valderrama Angel Jefferson
- Alejo Oviedo Carlos Roberto
- Muñoz Cordova Jose Bil Markus
- Leiva Luza Alexander David

CUSCO – PERÚ

2025

Etapa 1 - Ideación

Capítulo 1 – Fase de Ideación

1. Descripción del problema

El reto consiste en desarrollar un sistema que permita gestionar el árbol genealógico de una civilización ficticia utilizando un Árbol Binario de Búsqueda (ABB). El sistema debe permitir insertar, eliminar y buscar personas en el árbol, así como visualizar sus generaciones, ancestros y descendientes. Este proyecto tiene como objetivo aplicar estructuras de datos para organizar jerárquicamente la información genealógica y responder a distintas consultas y también de modelar jerárquicamente un árbol genealógico perteneciente a una civilización ficticia, representando las relaciones familiares de forma ordenada y eficiente.

2. Objetivo

Aplicar los conceptos de ABB para representar relaciones familiares jerárquicas, permitiendo la inserción, búsqueda, eliminación y visualización de miembros del árbol.

3. Requerimientos del sistema

- Funcionales
 - Insertar un nuevo miembro al árbol genealógico.
 - Eliminar un miembro existente.
 - Buscar un miembro por nombre.
 - Mostrar ancestros de un miembro.
 - Mostrar descendientes.
 - Mostrar el árbol por generación (niveles).
 - Recorridos inorden, preorden, postorden.
- No funcionales
 - El sistema debe ejecutarse en un entorno de consola (Dev C++).
 - El código debe seguir buenas prácticas de programación.
 - Debe ser fácil de usar para el usuario.
 - Eficiencia en operaciones de búsqueda y recorrido.

4. Respuesta a las preguntas guías

¿Qué información se debe almacenar en cada nodo del árbol?

Cada nodo del árbol representa a una persona, por lo tanto, debe almacenar:

- **Nombre de la persona** (tipo string)
- **Puntero al hijo izquierdo** (para miembros con nombre menor en orden alfabético)

- **Puntero al hijo derecho** (para miembros con nombre mayor en orden alfabético)

cpp

CopiarEditar

```
struct Persona {
    string nombre;
    Persona* izquierda;
    Persona* derecha;
};
```

¿Cómo insertar y eliminar miembros del árbol sin romper su estructura?

- **Insertar:** Se compara el nombre del nuevo miembro con el nodo actual. Si es menor, se inserta a la izquierda; si es mayor, a la derecha de una forma recursiva.

cpp

CopiarEditar

```
Persona* insertar(Persona* nodo, string nombre) {
    if (nodo == nullptr)
        return new Persona(nombre);
    if (nombre < nodo->nombre)
        nodo->izquierda = insertar(nodo->izquierda, nombre);
    else if (nombre > nodo->nombre)
        nodo->derecha = insertar(nodo->derecha, nombre);
    return nodo;
}
```

- **Eliminar:** En este código aún no se ha implementado, pero normalmente implica tres casos: el nodo es hoja, tiene un solo hijo, o tiene dos hijos (reemplazándolo con el mínimo del subárbol derecho).

¿Qué métodos permiten recorrer el árbol para visualizar la genealogía?

El árbol se puede recorrer con los siguientes métodos:

- **Inorden** (inorden): Muestra en orden alfabético.

- **Preorden** (preorden): Muestra primero el ancestro, luego descendientes.
- **Postorden** (postorden): Muestra primero descendientes y luego el ancestro.

Estos métodos permiten visualizar la genealogía desde diferentes perspectivas (generaciones, jerarquía, ramas).

¿Cómo determinar si un miembro pertenece a una rama específica?

Esto se puede hacer recorriendo el árbol desde la raíz, comparando el nombre del miembro buscado con los nodos hasta encontrarlo (búsqueda binaria) si se encuentra en la subrama izquierda, pertenece a esa rama, y lo mismo con la derecha.

Ejemplo de función (no incluida aún en tu código):

cpp

CopiarEditar

```
bool pertenece(Persona* nodo, string nombre) {
    if (!nodo) return false;
    if (nodo->nombre == nombre) return true;
    return nombre < nodo->nombre ?
        pertenece(nodo->izquierda, nombre) :
        pertenece(nodo->derecha, nombre);
}
```

¿Cómo balancear el árbol si se vuelve demasiado profundo?

Un árbol binario de búsqueda (ABB) puede volverse desequilibrado si los datos se insertan en orden para balancear se pueden usar:

- **Árboles AVL o Árboles Rojo-Negro**, que balancean automáticamente después de cada inserción/eliminación.
- En una solución simple, puedes recolectar los nodos en un array (mediante recorrido inorden) y reconstruir un árbol equilibrado desde ahí.

5. Estructura del ABB

Cada nodo del arbol contiene:

- Nombre del miembro
- Referencia al hijo izquierdo
- Referencia al hijo derecho

6. Operaciones Realizadas

- Insertar: Se insertan miembros ordenadamente segun orden alfabetico.
- Buscar: Se determina si un miembro existe en el arbol.
- Eliminar: Se remueve un miembro manteniendo la estructura del ABB.
- Recorridos: Visualizacion de la genealogia usando inorden, preorden y postorden.

7. Casos de Prueba

Miembros insertados en el ABB segun el siguiente arbol genealogico ficticio:

- Parte materna: Miriam (madre), Melchora (abuela), Pedro (abuelo)
- Parte paterna: Hugo (padre), Roberta (abuela), Leonidas (abuelo), Karina (tia), Sergio (tio), Walter (tio)
- Hermanos: Rodrigo

8. Justificación del Uso del ABB

Se eligió un ABB ya que permite mantener la información ordenada y facilita operaciones de búsqueda, inserción y eliminación en tiempo logarítmico en el mejor de los casos.

9. Resultados Obtenidos

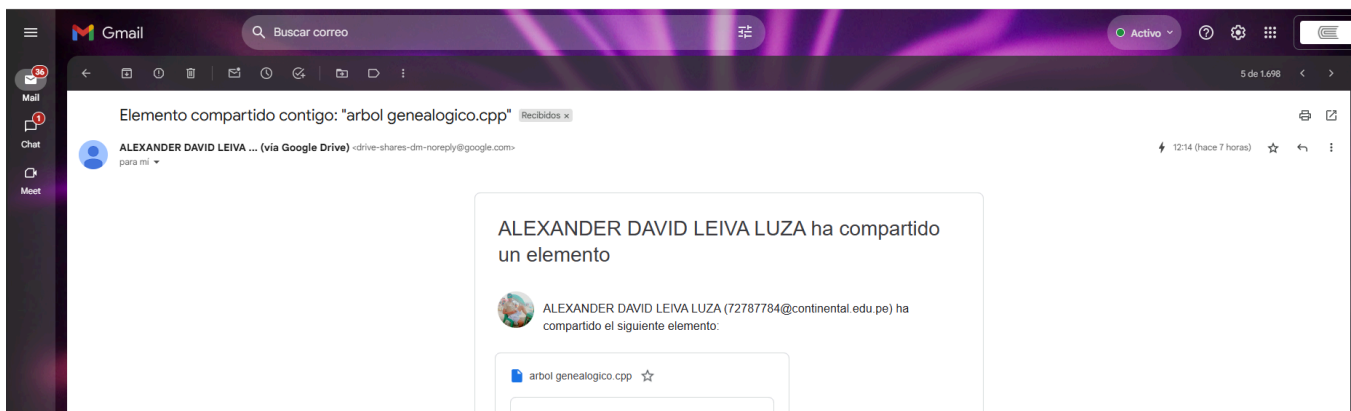
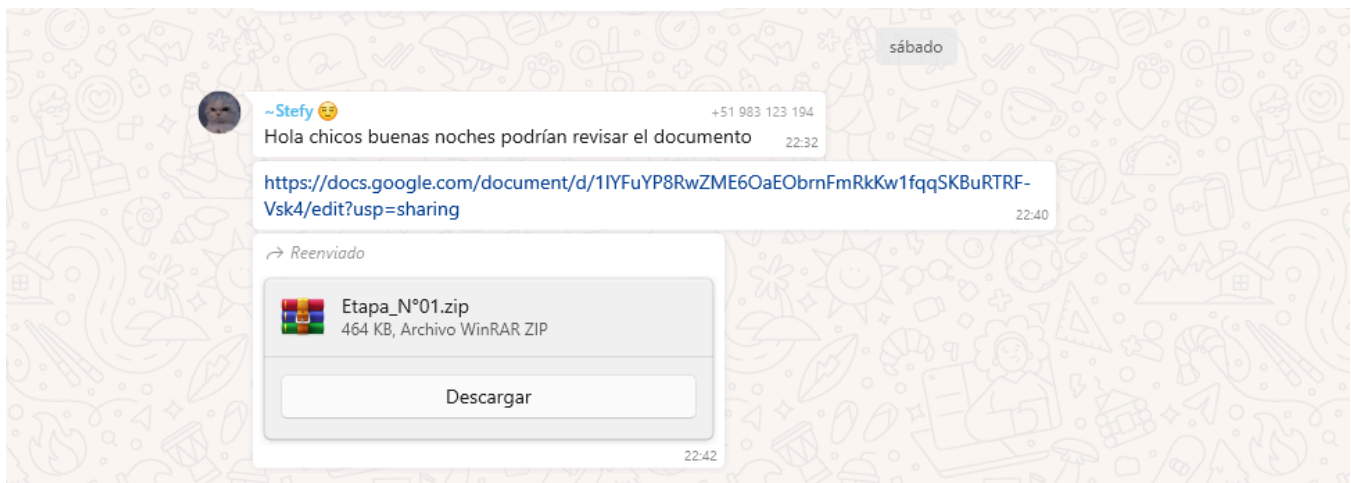
El sistema permite realizar correctamente todas las operaciones con múltiples pruebas y responde eficientemente a consultas sobre miembros.

10. Diagrama del Arbol Genealogico (en orden alfabetico)



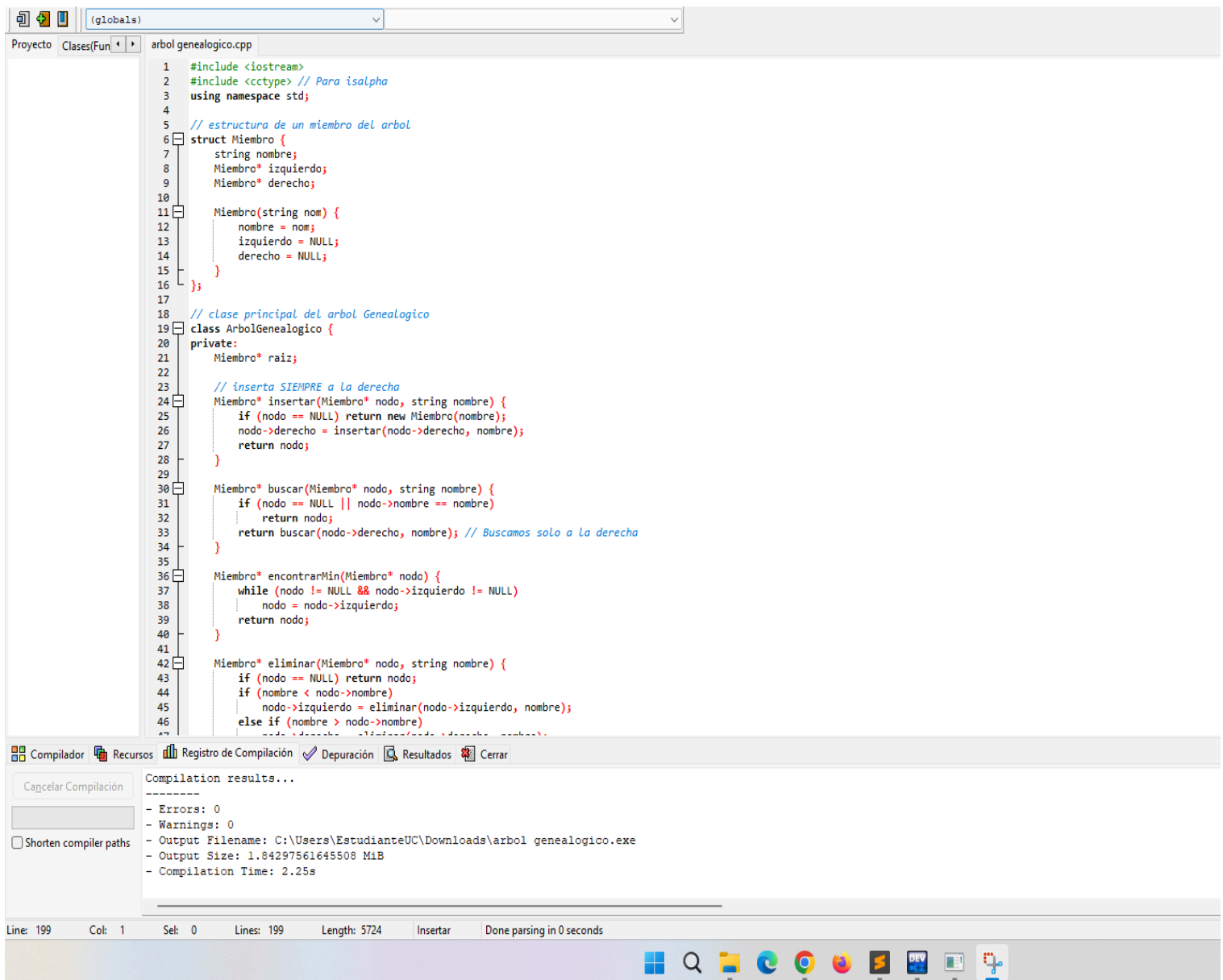
11. Herramienta Colaborativa (EVIDENCIA)

Uso de Whatsapp para coordinaciòn de trabajo y entrega por partes.



Solución:

- Diseño detallado del algoritmo para gestionar el árbol genealógico mediante un ABB, definiendo cómo se insertan, eliminan y consultan los miembros.
- Desarrollo de la lógica de recorrido del árbol (inorden, preorden, postorden) para distintos tipos de consulta (por generación, ancestros, descendientes).
- Codificación de la solución en Dev-C++, aplicando buenas prácticas de programación y guiándonos con lo aprendido con el docente en clases.



```
1 #include <iostream>
2 #include <cctype> // Para isalpha
3 using namespace std;
4
5 // estructura de un miembro del arbol
6 struct Miembro {
7     string nombre;
8     Miembro* izquierdo;
9     Miembro* derecho;
10
11     Miembro(string nom) {
12         nombre = nom;
13         izquierdo = NULL;
14         derecho = NULL;
15     }
16 };
17
18 // clase principal del arbol Genealogico
19 class ArbolGenealogico {
20 private:
21     Miembro* raiz;
22
23     // inserta SIEMPRE a la derecha
24     Miembro* insertar(Miembro* nodo, string nombre) {
25         if (nodo == NULL) return new Miembro(nombre);
26         nodo->derecho = insertar(nodo->derecho, nombre);
27         return nodo;
28     }
29
30     Miembro* buscar(Miembro* nodo, string nombre) {
31         if (nodo == NULL || nodo->nombre == nombre)
32             return nodo;
33         return buscar(nodo->derecho, nombre); // Buscamos solo a la derecha
34     }
35
36     Miembro* encontrarMin(Miembro* nodo) {
37         while (nodo != NULL && nodo->izquierdo != NULL)
38             nodo = nodo->izquierdo;
39         return nodo;
40     }
41
42     Miembro* eliminar(Miembro* nodo, string nombre) {
43         if (nodo == NULL) return nodo;
44         if (nombre < nodo->nombre)
45             nodo->izquierdo = eliminar(nodo->izquierdo, nombre);
46         else if (nombre > nodo->nombre)
47             nodo->derecho = eliminar(nodo->derecho, nombre);
48         else
49             // Eliminar nodo hoja
50             return NULL;
51     }
52
53 public:
54     ArbolGenealogico() { raiz = NULL; }
55     void insertar(string nombre) { insertar(raiz, nombre); }
56     Miembro* buscar(string nombre) { return buscar(raiz, nombre); }
57     void eliminar(string nombre) { eliminar(raiz, nombre); }
58     void mostrarPreorden() { mostrarPreorden(raiz); }
59     void mostrarInorden() { mostrarInorden(raiz); }
60     void mostrarPostorden() { mostrarPostorden(raiz); }
61 };
62
63 void mostrarPreorden(Miembro* nodo) {
64     if (nodo != NULL) {
65         cout << nodo->nombre << " ";
66         mostrarPreorden(nodo->izquierdo);
67         mostrarPreorden(nodo->derecho);
68     }
69 }
70
71 void mostrarInorden(Miembro* nodo) {
72     if (nodo != NULL) {
73         mostrarInorden(nodo->izquierdo);
74         cout << nodo->nombre << " ";
75         mostrarInorden(nodo->derecho);
76     }
77 }
78
79 void mostrarPostorden(Miembro* nodo) {
80     if (nodo != NULL) {
81         mostrarPostorden(nodo->izquierdo);
82         mostrarPostorden(nodo->derecho);
83         cout << nodo->nombre << " ";
84     }
85 }
```

Compilation results...

- Errors: 0

- Warnings: 0

- Output Filename: C:\Users\EstudienteUC\Downloads\arbol genealogico.exe

- Output Size: 1.84297561645508 MiB

- Compilation Time: 2.25s

Line: 199 Col: 1 Sel: 0 Lines: 199 Length: 5724 Insertar Done parsing in 0 seconds

Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Vgntana Ayuda

TM-GCC 4.9.2 64-bit Release

Proyecto Clases(Fun) arbol genealogico.cpp

```
40
41
42 Miembro* eliminar(Miembro* nodo, string nombre) {
43     if (nodo == NULL) return nodo;
44     if (nombre < nodo->nombre)
45         nodo->izquierdo = eliminar(nodo->izquierdo, nombre);
46     else if (nombre > nodo->nombre)
47         nodo->derecho = eliminar(nodo->derecho, nombre);
48     else {
49         if (nodo->izquierdo == NULL) {
50             Miembro* temp = nodo->derecho;
51             delete nodo;
52             return temp;
53         } else if (nodo->derecho == NULL) {
54             Miembro* temp = nodo->izquierdo;
55             delete nodo;
56             return temp;
57         }
58         Miembro* temp = encontrarMin(nodo->derecho);
59         nodo->nombre = temp->nombre;
60         nodo->derecho = eliminar(nodo->derecho, temp->nombre);
61     }
62     return nodo;
63 }
64
65 void inorden(Miembro* nodo) {
66     if (nodo == NULL) return;
67     inorden(nodo->izquierdo);
68     cout << nodo->nombre << " ";
69     inorden(nodo->derecho);
70 }
71
72 void preorden(Miembro* nodo) {
73     if (nodo == NULL) return;
74     cout << nodo->nombre << " ";
75     preorden(nodo->izquierdo);
76     preorden(nodo->derecho);
77 }
78
79 void postorden(Miembro* nodo) {
80     if (nodo == NULL) return;
81     postorden(nodo->izquierdo);
82     postorden(nodo->derecho);
83     cout << nodo->nombre << " ";
84 }
85
```

Compilador Recursos Registro de Compilación Depuración Resultados Cerrar

Compilation results...

Errors: 0

Warnings: 0

Output Filename: C:\Users\EstudianteUC\Downloads\arbol genealogico.exe

Output Size: 1.84297561645508 MiB

Compilation Time: 2.25s

Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Vgntana Ayuda

TM-GCC 4.9.2 64-bit Release

Proyecto Clases(Fun) arbol genealogico.cpp

```
82     postorden(nodo->derecho);
83     cout << nodo->nombre << " ";
84 }
85
86 public:
87 ArbolGenealogico() {
88     raiz = NULL;
89 }
90
91 void insertar(string nombre) {
92     raiz = insertar(raiz, nombre);
93 }
94
95 bool buscar(string nombre) {
96     return buscar(raiz, nombre) != NULL;
97 }
98
99 void eliminar(string nombre) {
100     raiz = eliminar(raiz, nombre);
101 }
102
103 void mostrarInorden() {
104     cout << "Inorden: ";
105     inorden(raiz);
106     cout << endl;
107 }
108
109 void mostrarPreorden() {
110     cout << "Preorden: ";
111     preorden(raiz);
112     cout << endl;
113 }
114
115 void mostrarPostorden() {
116     cout << "Postorden: ";
117     postorden(raiz);
118     cout << endl;
119 }
120 };
121
122 // Función para validar nombres (solo letras sin espacios)
123 bool nombreValido(const string& nombre) {
124     if (nombre.empty()) return false;
125     for (char c : nombre) {
126         if (!isalpha(c)) return false;
127     }
128     return true;
129 }
```

Compilador Recursos Registro de Compilación Depuración Resultados Cerrar

Compilation results...

Errors: 0

Warnings: 0

Output Filename: C:\Users\EstudianteUC\Downloads\arbol genealogico.exe

Output Size: 1.84297561645508 MiB

Compilation Time: 2.25s

Line: 199 Col: 1 Sel: 0 Lines: 199 Length: 5724 Insertar Done parsing in 0 seconds

C:\Users\EstudienteUC\Downloads\arbol genealogico.cpp - [Executing] - Dev-C++ 5.11

Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Ventana Ayuda

TDM-GCC 4.9.2 64-bit Release

(globals)

Proyecto Clases(Fun) arbol genealogico.cpp

```
121
122 // Función para validar nombres (solo letras sin espacios)
123 bool nombreValido(const string& nombre) {
124     if (nombre.empty()) return false;
125     for (char c : nombre) {
126         if (!isalpha(c)) return false;
127     }
128     return true;
129 }
130
131 // menú principal
132 int main() {
133     ArbolGenealogico arbol;
134     int opcion;
135     string nombre;
136
137     do {
138         cout << "\n=== MENU DEL ARBOL GENEALOGICO ===\n";
139         cout << "1. Insertar miembros\n";
140         cout << "2. Buscar miembro\n";
141         cout << "3. Eliminar miembro\n";
142         cout << "4. Mostrar Inorden\n";
143         cout << "5. Mostrar Preorden\n";
144         cout << "6. Mostrar Postorden\n";
145         cout << "7. Salir\n";
146         cout << "Seleccione una opcion: ";
147         cin >> opcion;
148
149         switch(opcion) {
150             case 1: {
151                 int cantidad;
152                 cout << "¿Cuántos miembros desea insertar?: ";
153                 cin >> cantidad;
154                 for (int i = 0; i < cantidad; i++) {
155                     do {
156                         cout << "Ingrese el nombre del miembro #" << i + 1 << ": ";
157                         cin >> nombre;
158                         if (!nombreValido(nombre))
159                             cout << "Nombre invalido. Use solo letras sin espacios.\n";
160                         while (!nombreValido(nombre));
161                         arbol.insertar(nombre);
162                     }
163                     cout << "Miembros insertados con exito.\n";
164                     break;
165                 }
166             case 2: {
```

Compilador Recursos Registro de Compilación Depuración Resultados Cerrar

Cancelar Compilación

☐ Shorten compiler paths

Compilation results...

```
-----
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\EstudienteUC\Downloads\arbol genealogico.exe
- Output Size: 1.84297561645508 MiB
- Compilation Time: 2.25s
```

Line: 199 Col: 1 Sel: 0 Lines: 199 Length: 5724 Insertar Done parsing in 0 seconds



```
(globals)
oyecto Clases(Fun < > > arbol genealogico.cpp
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199

for (int i = 0; i < cantidad; i++) {
    do {
        cout << "Ingrese el nombre del miembro #" << i + 1 << ": ";
        cin >> nombre;
        if (!nombreValido(nombre))
            cout << "Nombre invalido. Use solo letras sin espacios.\n";
        } while (!nombreValido(nombre));
        arbol.insertar(nombre);
    }
    cout << "Miembros insertados con exito.\n";
    break;
}

case 2:
    cout << "Ingrese el nombre a buscar: ";
    cin >> nombre;
    if (arbol.buscar(nombre))
        cout << nombre << " SI existe en el arbol.\n";
    else
        cout << nombre << " NO se encuentra.\n";
    break;

case 3:
    cout << "Ingrese el nombre a eliminar: ";
    cin >> nombre;
    arbol.eliminar(nombre);
    cout << "Miembro eliminado (si existia).\n";
    break;

case 4:
    arbol.mostrarInorden();
    break;

case 5:
    arbol.mostrarPreorden();
    break;

case 6:
    arbol.mostrarPostorden();
    break;

case 7:
    cout << "Saliendo del programa.\n";
    break;

default:
    cout << "Opcion invalida. Intente de nuevo.\n";
}

} while (opcion != 7);

return 0;
}
```

Compilador Recursos Registro de Compilación Depuración Resultados Cerrar

Cancelar Compilación Shorten compiler paths

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\EstudienteUC\Downloads\arbol genealogico.exe
- Output Size: 1.84297561645508 MiB
- Compilation Time: 2.25s

e: 199 Col: 1 Sel: 0 Lines: 199 Length: 5724 Insertar Done parsing in 0 seconds

```
C:\Users\EstudienteUC\Downloads\arbol genealogico.exe
3. Eliminar miembro
4. Mostrar Inorden
5. Mostrar Preorden
6. Mostrar Postorden
7. Salir
Seleccione una opcion: 6
Postorden: angel daya

=== MENU DEL ARBOL GENEALOGICO ===
1. Insertar miembros
2. Buscar miembro
3. Eliminar miembro
4. Mostrar Inorden
5. Mostrar Preorden
6. Mostrar Postorden
7. Salir
Seleccione una opcion:
3
Ingrese el nombre a eliminar: daya
Miembro eliminado (si existia).

=== MENU DEL ARBOL GENEALOGICO ===
1. Insertar miembros
2. Buscar miembro
3. Eliminar miembro
4. Mostrar Inorden
5. Mostrar Preorden
6. Mostrar Postorden
7. Salir
Seleccione una opcion:
```

CÓDIGO

```
#include <iostream>

using namespace std;

struct Miembro {
    string nombre;
    Miembro* izquierdo;
    Miembro* derecho;
    Miembro(string nom) {
        nombre = nom;
        izquierdo = NULL;
        derecho = NULL;
    }
};

class ArbolGenealogico {
private:

    Miembro* raiz;

    Miembro* insertar(Miembro* nodo, string nombre) {
        if (nodo == NULL) return new Miembro(nombre);
        if (nombre < nodo->nombre)
            nodo->izquierdo = insertar(nodo->izquierdo, nombre);
        else if (nombre > nodo->nombre)
            nodo->derecho = insertar(nodo->derecho, nombre);
        return nodo;
    }

    Miembro* buscar(Miembro* nodo, string nombre) {
```

```

if (nodo == NULL || nodo->nombre == nombre)
return nodo;

if (nombre < nodo->nombre)
return buscar(nodo->izquierdo, nombre);
else
return buscar(nodo->derecho, nombre);
}

Miembro* encontrarMin(Miembro* nodo) {
while (nodo != NULL && nodo->izquierdo != NULL)
nodo = nodo->izquierdo;
return nodo;
}

Miembro* eliminar(Miembro* nodo, string nombre) {
if (nodo == NULL) return nodo;
if (nombre < nodo->nombre)
nodo->izquierdo = eliminar(nodo->izquierdo, nombre);
else if (nombre > nodo->nombre)
nodo->derecho = eliminar(nodo->derecho, nombre);
else {
if (nodo->izquierdo == NULL) {
Miembro* temp = nodo->derecho;
delete nodo;
return temp;
} else if (nodo->derecho == NULL) {
Miembro* temp = nodo->izquierdo;
delete nodo;

```

```

return temp;

}

Miembro* temp = encontrarMin(nodo->derecho);

nodo->nombre = temp->nombre;

nodo->derecho = eliminar(nodo->derecho, temp->nombre);

}

return nodo;

}

void inorden(Miembro* nodo) {
if (nodo == NULL) return;
inorden(nodo->izquierdo);
cout << nodo->nombre << " ";
inorden(nodo->derecho);
}

void preorden(Miembro* nodo) {
if (nodo == NULL) return;
cout << nodo->nombre << " ";
preorden(nodo->izquierdo);
preorden(nodo->derecho);
}

void postorden(Miembro* nodo) {
if (nodo == NULL) return;
postorden(nodo->izquierdo);
postorden(nodo->derecho);
cout << nodo->nombre << " ";
}

```

```

}

public:
ArbolGenealogico() {
    raiz = NULL;
}

void insertar(string nombre) {
    raiz = insertar(raiz, nombre);
}

bool buscar(string nombre) {
    return buscar(raiz, nombre) != NULL;
}

void eliminar(string nombre) {
    raiz = eliminar(raiz, nombre);
}

void mostrarInorden() {
    cout << "Inorden: ";
    inorden(raiz);
    cout << endl;
}

void mostrarPreorden() {
    cout << "Preorden: ";
    preorden(raiz);
    cout << endl;
}

void mostrarPostorden() {

```

```
cout << "Postorden: ";
postorden(raiz);
cout << endl;
}
};

int main() {
ArbolGenealogico arbol;

int opcion;

string nombre;

do {
cout << "\n=== MENU DEL ARBOL GENEALOGICO ===\n";
cout << "1. Insertar miembro\n";
cout << "2. Buscar miembro\n";
cout << "3. Eliminar miembro\n";
cout << "4. Mostrar Inorden\n";
cout << "5. Mostrar Preorden\n";
cout << "6. Mostrar Postorden\n";
cout << "0. Salir\n";
cout << "Seleccione una opcion: ";

cin >> opcion;

switch(opcion) {

case 1:

cout << "Ingrese el nombre del miembro a insertar: ";

cin >> nombre;

arbol.insertar(nombre);

cout << "Miembro insertado con exito.\n";
```

```
break;

case 2:

cout << "Ingrese el nombre a buscar: ";

cin >> nombre;

if (arbol.buscar(nombre))

cout << nombre << " si existe en el arbol.\n";

else

cout << nombre << " no se encuentra.\n";

break;

case 3:

cout << "Ingrese el nombre a eliminar: ";

cin >> nombre;

arbol.eliminar(nombre);

cout << "Miembro eliminado (si existia).\n";

break;

case 4:

arbol.mostrarInorden();

break;

case 5:

arbol.mostrarPreorden();

break;

case 6:

arbol.mostrarPostorden();

break;

case 0:
```



```
cout << "Saliendo del programa.\n";  
break;  
default:  
cout << "Opcion invalida. Intente de nuevo.\n";  
}  
} while (opcion != 0);  
return 0;  
}
```