



ABR-ÁRBOL GENEALÓGICO

ESTRUCTURA DE DATOS

Docente:

Rosario Delia Osorio Contreras





INTEGRANTES



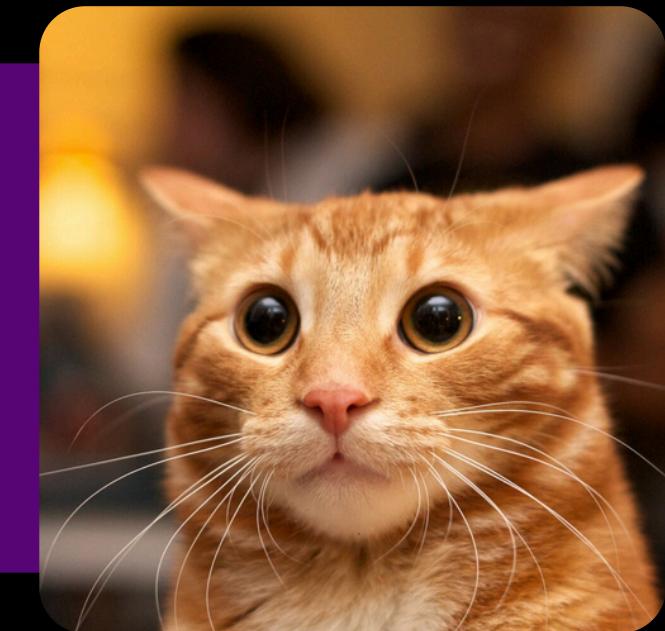
Merino Abarca
NIDIA EVELYN



Paitan Rios
SHAMIRA SHANTAL



Muños de la Cruz
MARCO ANTONIO



Chuquimantari Avila
JAIRO SAM





OBJETIVO DEL PROGRAMA

Digitalizar, representar y manejar el legado o linaje de una familia o civilización digitalmente, con la capacidad de mantener ordenados y actualizar los datos de sus miembros, facilitando la consulta y manipulación de esa información.



Nodo



Podemos verlo como una "caja" que guarda datos y conexiones a otras cajas.

Es fundamental en estructuras como árboles, donde cada nodo puede tener hijos que forman una jerarquía.

Estructura básica de la estructura de datos, que en un árbol binario nos permite:

- Información (como nombre, ID, fecha, etc.).
- Un puntero al hijo izquierdo.
- Un puntero al hijo derecho.



Estructura nodo

En esta parte del código cada nodo del árbol representa a una persona:

Tiene datos básicos:

- nombre: nombre de la persona.
- id: un identificador único.
- fechaNacimiento: fecha de nacimiento (como texto).

Tiene tres punteros:

- Nodo Izquierdo: apunta al primer hijo.
- Nodo Derecho: apunta al segundo hijo.
- Nodo padre: Apunta al padre del nodo (de donde proviene).

Inicialmente, los punteros a los hijos son NULL, es decir, el nodo no tiene hijos.

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 struct Nodo {
6     string nombre;
7     string id;
8     string fechaNacimiento;
9     Nodo* izquierdo;
10    Nodo* derecho;
11    Nodo* padre;
12
13    Nodo(string _nombre, string _id, string _fechaNacimiento) {
14        nombre = _nombre;
15        id = _id;
16        fechaNacimiento = _fechaNacimiento;
17        izquierdo = NULL;
18        derecho = NULL;
19        padre = NULL;
20    }
21}
```



INSERTAR UN NODO

Es una función recursiva que inserta un nuevo nodo ordenandolo por el nombre.

En caso de que el nombre es menor al del nodo actual, se va al subárbol izquierdo; si es mayor o igual, al derecho.

Además de insertar, también guarda quién es el *padre* del nodo insertado.

```
Nodo* insertar(Nodo* raiz, Nodo* nuevo, Nodo* padre = NULL) {
    if (raiz == NULL) {
        nuevo->padre = padre;
        return nuevo;
    }
    if (nuevo->nombre < raiz->nombre) {
        raiz->izquierdo = insertar(raiz->izquierdo, nuevo, raiz);
    } else {
        raiz->derecho = insertar(raiz->derecho, nuevo, raiz);
    }
    return raiz;
}
```

Recorrido Inorden

Cuando el árbol está ordenado (como un árbol binario de búsqueda), el recorrido inorden imprime los nodos en orden creciente según la clave usada para ordenar. En tu código, aunque insertas ordenando por nombre, el árbol está estructurado de forma que el recorrido inorden lista los miembros por nombre en orden alfabético.

Es decir:
izquierda-> nodo → derecha



El recorrido inorden visita:
primero el subárbol izquierdo:

(Ana)

Después el nodo actual: (Marta)
Finalmente el subárbol derecho:
(Luis)

El resultado sería este:

Ana - id - fechaNacimiento
Marta - id - fechaNacimiento
Luis - id - fechaNacimiento



Recorrido Inorden



Muestra los nodos en orden alfabético por nombre.

Se hace así:

- Recorrer el subárbol izquierdo (todos los nodos del lado izquierdo)
- Visitar el nodo actual (imprimir o procesar sus datos)
- Recorrer el subárbol derecho (todos los nodos del lado derecho)

Visita izquierda,
Imprime actual,
Visita derecha.

```
void inorden(Nodo* nodo) {
    if (nodo != NULL) {
        inorden(nodo->izquierdo);
        cout << nodo->nombre << " - " << nodo->id << " - " << nodo->fechaNacimiento << endl;
        inorden(nodo->derecho);
    }
}
```



Buscar un miembro

Busca recursivamente un miembro por ID:

Si encuentra el ID: retorna el nodo.

Si el ID buscado es menor al actual, va a la izquierda.

Si es mayor, va a la derecha.

Aquí hay un problema importante: la búsqueda por ID no funciona correctamente porque el árbol está ordenado por nombre, no por ID. Entonces, la comparación if ($id < \text{raiz-} \rightarrow id$) no tiene sentido y podría buscar incorrectamente.

```
Nodo* encontrarMinimo(Nodo* nodo) {  
    while (nodo->izquierdo != NULL) {  
        nodo = nodo->izquierdo;  
    }  
    return nodo;
```



Eliminar miembro

También basado en búsqueda por ID (con el mismo problema que arriba).

Funciona así:

Si el nodo tiene un solo hijo, se reemplaza por ese hijo.

Si tiene dos hijos, se reemplaza por el nodo mínimo del subárbol derecho (técnica clásica).

Si es hoja, simplemente se elimina.



```
Nodo* eliminar(Nodo* raiz, const string& id) {
    if (raiz == NULL) return NULL;

    if (id < raiz->id) {
        raiz->izquierdo = eliminar(raiz->izquierdo, id);
    } else if (id > raiz->id) {
        raiz->derecho = eliminar(raiz->derecho, id);
    } else {
        if (raiz->izquierdo == NULL) {
            Nodo* temp = raiz->derecho;
            delete raiz;

            return temp;
        } else {
            Nodo* temp = encontrarMinimo(raiz->derecho);
            raiz->nombre = temp->nombre;
            raiz->id = temp->id;
            raiz->fechaNacimiento = temp->fechaNacimiento;
            raiz->derecho = eliminar(raiz->derecho, temp->id);
        }
    }
    return raiz;
}
```



Actualizar miembro

Llama a buscar para encontrar el nodo con ese ID.

Si lo encuentra, pide por consola un nuevo nombre y fecha de nacimiento.

Actualiza esos valores en el nodo.



```
void actualizar(Nodo* raiz, const string& id) {
    Nodo* nodo = buscar(raiz, id);
    if (nodo == NULL) {
        cout << "Miembro no encontrado.\n";
        return;
    }
    cout << "Miembro encontrado: " << nodo->nombre << " - " << nodo->fechaNacimiento << endl;
    cout << "Nuevo nombre: ";
    getline(cin, nodo->nombre);
    cout << "Nueva fecha de nacimiento (YYYY-MM-DD): ";
    getline(cin, nodo->fechaNacimiento);
    cout << "Datos actualizados correctamente.\n";
}
```

Buscar por id



Busca un nodo por su id recorriendo todo el árbol , porque id no está ordenado. Así puedes encontrar cualquier miembro con su id.



```
Nodo* buscarPorID(Nodo* raiz, const string& id) {  
    if (raiz == NULL) return NULL;  
    if (raiz->id == id) return raiz;  
  
    Nodo* encontradoIzq = buscarPorID(raiz->izquierdo, id);  
    if (encontradoIzq != NULL) return encontradoIzq;  
  
    return buscarPorID(raiz->derecho, id);  
}
```



Mostrar ancestros

- Primero revisa si el nodo (miembro) existe. Si no, dice que no lo encontró.
- Si el nodo no tiene padre (es el ancestro más viejo que hay), dice que no tiene ancestros registrados.
- Si tiene padre, empieza desde ese padre y sigue subiendo mientras haya padre, mostrando en pantalla el nombre, id y fecha de cada uno.

```
void mostrarAncestros(Nodo* nodo) {  
    if (nodo == NULL) {  
        cout << "Miembro no encontrado.\n";  
        return;  
    }  
    if (nodo->padre == NULL) {  
        cout << nodo->nombre << " no tiene ancestros registrados.\n";  
        return;  
    }  
    cout << "Ancestros de " << nodo->nombre << ":\n";  
    Nodo* actual = nodo->padre;  
    while (actual != NULL) {  
        cout << actual->nombre << " - " << actual->id << " - " << actual->fechaNacimiento << endl;  
        actual = actual->padre;  
    }  
}
```

Mostrar Hijos



- Primero revisa si el nodo existe. Si no, avisa que no encontró al miembro.
- Revisa si tiene hijo izquierdo y lo muestra.
- Revisa si tiene hijo derecho y lo muestra.
- Si no tiene ninguno, dice que no tiene hijos registrados.



```
void mostrarHijos(Nodo* nodo) {
    if (nodo == NULL) {
        cout << "Miembro no encontrado.\n";
        return;
    }
    cout << "Hijos de " << nodo->nombre << ":\n";
    bool tieneHijos = false;
    if (nodo->izquierdo != NULL) {
        cout << "- " << nodo->izquierdo->nombre << " - " << nodo->izquierdo->id << " - " << nodo->izquierdo->fechaNacimiento << endl;
        tieneHijos = true;
    }
    if (nodo->derecho != NULL) {
        cout << "- " << nodo->derecho->nombre << " - " << nodo->derecho->id << " - " << nodo->derecho->fechaNacimiento << endl;
        tieneHijos = true;
    }
    if (!tieneHijos) {
        cout << "No tiene hijos registrados.\n";
    }
}
```

GRACIAS

