

Árbol Genealógico

Curso: Estructura de datos

Integrantes:

- Huayllani Otarola Erick Marvil
- Balbin Gutierrez Leonardo
- Torres Escalante Jhon Jaime

Define una estructura llamada "Persona" en C++ ,que está diseñada para construir una estructura de datos jerárquica o árbol.

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  using namespace std;
5
6  struct Persona {
7      string nombre;
8      int edad;
9      Persona* padre;
10     Persona* izquierda;
11     Persona* derecha;
12 }
```

1. Usamos un string para almacenar el nombre de la **persona**.
2. **En padre**:Un puntero a otra Persona (el padre en una jerarquía familiar o en un árbol).
3. La **"Izquierda"** un puntero a otra persona(de un Hij@ izquierdo en la estructura del árbol).
4. La **"Derecha"** un puntero a otra persona(de un hij@ derecho en el árbol binario)

Implementa una función para crear y gestionar instancias de la estructura Persona en un árbol binario.

```
14  vector<Persona*> personas;
15
16  Persona* crearPersona(string nombre, int edad) {
17      Persona* nueva = new Persona;
18      nueva->nombre = nombre;
19      nueva->edad = edad;
20      nueva->padre = NULL;
21      nueva->izquierda = NULL;
22      nueva->derecha = NULL;
23      personas.push_back(nueva);
24      return nueva;
25  }
```

1.Registro Central:

- Vector global que almacena todos los nodos creados
- Permite seguimiento de todas las personas en el árbol

2.Inicialización Segura:

- Todos los punteros se inicializan a NULL
- Evita referencias no válidas

3.Gestión de Memoria:

- Uso de “new” para asignación dinámica
- Devuelve puntero para construcción del árbol

Implementamos una función para buscar Personas por nombre en el registro.

Declaramos la función

```
27 Persona* buscarPersona(string nombre) {  
28     for (int i = 0; i < personas.size(); i++) {  
29         if (personas[i]->nombre == nombre)  
30             return personas[i];  
31     }  
32     return NULL;  
33 }
```

comparación de nombres y retorno si encuentra y ala vez si no encuentra-

1.Mecanismo de Búsqueda:

- Recorre el vector personas de forma secuencial.
- Compara el nombre buscado con cada elemento.
- Retorna al primer match encontrado.

Va permitir establecer relaciones padre-hijo en una estructura de árbol binario.

```
35 void agregarHijo(Persona* padre, Persona* hijo) {  
36     if (padre->izquierda == NULL) {  
37         padre->izquierda = hijo;  
38     } else if (padre->derecha == NULL) {  
39         padre->derecha = hijo;  
40     } else {  
41         cout << "El padre ya tiene dos hij@s.\n";  
42         return;  
43     }  
44     hijo->padre = padre;  
45 }
```

1. Realizará la declaración.
2. Verifica y asigna hijo izquierdo.
3. Verifica y asigna hijo derecho.

Advertencia de datos existentes y salida temprana

Establece relación inversa

1.Gestión de Relaciones:

- Establece relación bidireccional(Permite navegar el árbol en ambas direcciones).
- Actualiza puntero padre del hijo automáticamente.

Muestra todos los ancestros de una persona (recursivamente hacia arriba) y Muestra todos los descendientes de una persona (recursivamente hacia abajo)

```
47 void mostrarAncestros(Persona* persona) {
48     if (persona->padre != NULL) {
49         cout << persona->padre->nombre << " (" << persona->padre->edad << ")\n";
50         mostrarAncestros(persona->padre);
51     }
52 }
53
54 void mostrarDescendientes(Persona* persona) {
55     if (persona != NULL) {
56         if (persona->izquierda)
57             cout << persona->izquierda->nombre << " (" << persona->izquierda->edad << ")\n";
58         if (persona->derecha)
59             cout << persona->derecha->nombre << " (" << persona->derecha->edad << ")\n";
60         mostrarDescendientes(persona->izquierda);
61         mostrarDescendientes(persona->derecha);
62     }
63 }
64
```

1. **Recursión simple:** Solo avanza hacia el padre
2. **Formato:** Muestra "Nombre (Edad)"
3. **Orden:** Del padre directo al ancestro más lejano
4. **Limitación:** No verifica si persona es "NULL" inicialmente.

Recursividad en ambos árboles

1. **Recursión doble:** Va explora ambos hijos
2. **Orden:** Primero hijos directos, luego sus descendientes
3. **Seguridad:** Verifica "NULL" antes de acceder a punteros
4. **Formato consistente:** Igual que ancestros ("Nombre (Edad)")

La función que elimina recursivamente un subárbol completo desde un nodo dado, incluyendo(Todos sus descendientes, Él propio nodo, Su referencia en el vector global personas)

```
65 void eliminarSubarbol(Persona* persona) {  
66     if (persona == NULL) return;  
67  
68     eliminarSubarbol(persona->izquierda);  
69     eliminarSubarbol(persona->derecha);  
70  
71     for (int i = 0; i < personas.size(); i++) {  
72         if (personas[i] == persona) {  
73             personas.erase(personas.begin() + i);  
74             break;  
75         }  
76     }  
77     delete persona;  
78 }
```

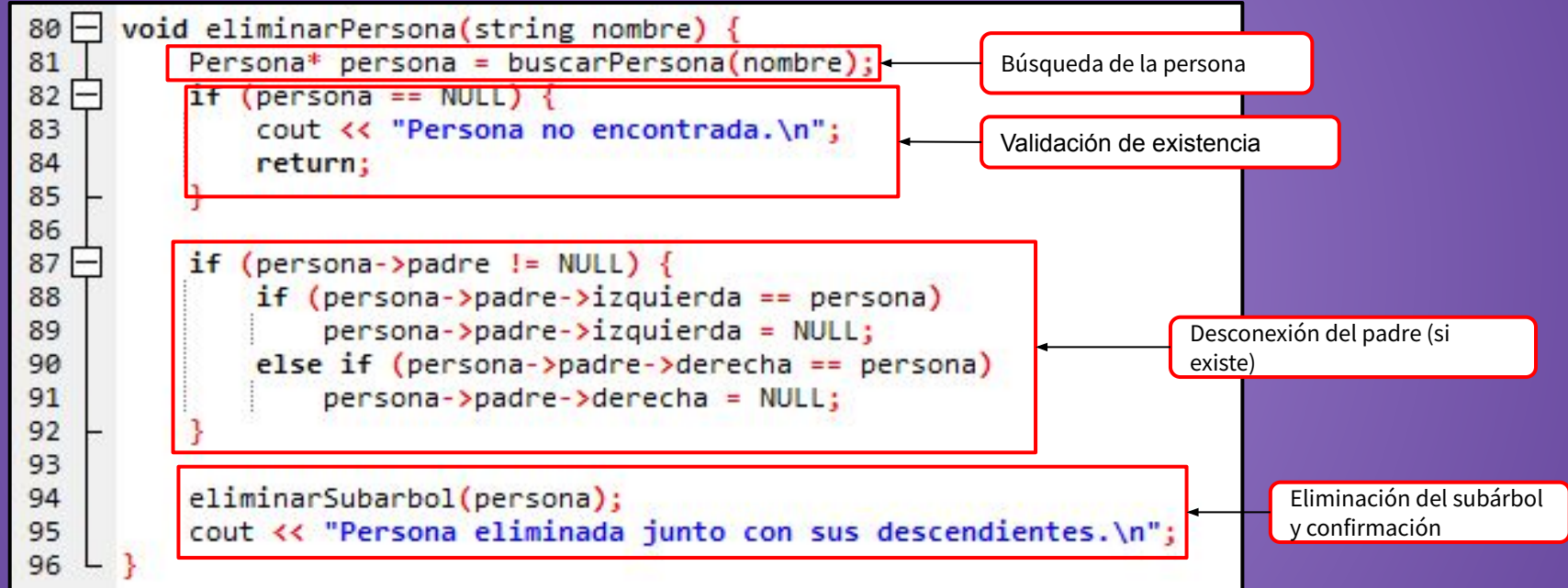
Eliminación recursiva de hijos

Eliminación del registro en el vector global

1. Eliminación en profundidad (post-order):

- Primero elimina hijos izquierdo y derecho (recursivamente).
- Luego elimina el nodo actual.

La función elimina una persona y todo su subárbol de descendientes, manteniendo la integridad de la estructura del árbol.



Una función que verifica y solicita al usuario seleccionar una opción mediante un número del (1 al 6), asegurándose de gestionar de manera efectiva cualquier error o entrada inválida.

```
98 int obtenerOpcionValida() {
99     int opcion;
100     while (true) {
101         cout << "Opcion: ";
102         if (cin >> opcion) {
103             if (opcion >= 1 && opcion <= 6) {
104                 cin.ignore(); // Limpiar el buffer(memori almacenada)
105                 return opcion;
106             } else {
107                 cout << "Por favor elija una opcion valida del menu (1-6)...\\n";
108             }
109         } else {
110             cout << "Entrada invalida. Por favor ingrese un numero del menu...\\n";
111             cin.clear(); // Limpiar el estado de error
112             while (cin.get() != '\\n'); // Limpia el buffer(memori almacenada)
113         }
114     }
115 }
```

Interfaz amigable:

1. Detecta entradas no numéricas
2. Recupera el flujo de entrada después de errores
3. Limpieza completa del buffer cuando hay error
4. Mensajes claros para el usuario
5. Bucle continuo hasta obtener entrada válida

La función solicita y verifica que la edad ingresada por el usuario sea un número no negativo, incluyendo un manejo adecuado de errores para entradas incorrectas o inválidas.

```
117 int obtenerEdadValida() {  
118     int edad;  
119     while (true) {  
120         cout << "Edad: ";  
121         if (cin >> edad) {  
122             if (edad >= 0) {  
123                 cin.ignore(); // Limpiar el buffer(memori almacenada)  
124                 return edad;  
125             } else {  
126                 cout << "La edad no puede ser negativa. Por favor ingrese un valor valido.\n";  
127             }  
128         } else {  
129             cout << "Entrada invalida. Por favor ingrese un numero valido.\n";  
130             cin.clear();  
131             while (cin.get() != '\n');  
132         }  
133     }  
134 }
```

Validación estricta:

1. Solo acepta valores enteros no negativos (edad ≥ 0)

-Manejo robusto de errores:

2. Detecta entradas no numéricas (ejemplo: "abc")

-Rechaza valores negativos

-Recupera el flujo de entrada tras errores

3. Limpieza de buffer:

- "cin.ignore()" tras lectura exitosa

-Limpieza completa con "cin.clear()" + "bucle" cuando hay error

4. Interfaz clara:

-Mensajes de error específicos

-Bucle continuo hasta entrada válida

```

136 int main() {
137     int opcion;
138     string nombre, nombrePadre;
139     int edad;
140     Persona* raiz = NULL;
141
142     do {
143         cout << "\n--- Menu de Arbol Genealogico ---\n";
144         cout << "1. Agregar persona(s)\n";
145         cout << "2. Establecer relacion padre-hijo\n";
146         cout << "3. Mostrar ancestros\n";
147         cout << "4. Mostrar descendientes\n";
148         cout << "5. Eliminar persona\n";
149         cout << "6. Salir\n";
150
151         opcion = obtenerOpcionValida();
152
153         if (opcion == 1) {
154             do {
155                 cout << "\nIngrese el nombre (o 0 para terminar de ingresar datos): ";
156                 getline(cin, nombre);
157
158                 if (nombre == "0") break;
159
160                 edad = obtenerEdadValida();
161
162                 Persona* nueva = crearPersona(nombre, edad);
163                 if (raiz == NULL) raiz = nueva;
164                 cout << "Persona registrada.\n";
165
166             } while (true);
167
168         } else if (opcion == 2) {
169             cout << "Nombre del padre: ";
170             getline(cin, nombrePadre);
171             Persona* padre = buscarPersona(nombrePadre);
172             if (!padre) {
173                 cout << "Padre no encontrado.\n";
174                 continue;
175             }
176             cout << "Nombre del hijo: ";
177             getline(cin, nombre);
178             Persona* hijo = buscarPersona(nombre);
179             if (!hijo) {
180                 cout << "Hijo no encontrado.\n";
181                 continue;
182             }
183             agregarHijo(padre, hijo);
184         } else if (opcion == 3) {
185             cout << "Nombre: ";
186             getline(cin, nombre);
187             Persona* persona = buscarPersona(nombre);
188             if (!persona) cout << "No encontrada.\n";
189             else mostrarAncestros(persona);

```

```

190         } else if (opcion == 4) {
191             cout << "Nombre: ";
192             getline(cin, nombre);
193             Persona* persona = buscarPersona(nombre);
194             if (!persona) cout << "No encontrada.\n";
195             else mostrarDescendientes(persona);
196         } else if (opcion == 5) {
197             cout << "Nombre de la persona a eliminar: ";
198             getline(cin, nombre);
199             eliminarPersona(nombre);
200         }
201     } while (opcion != 6);
202
203     cout << "Cerrando el sistema.Hasta la proxima[Presionar enter]...\n";
204     return 0;
205 }
206

```

Menú interactivo para gestionar un árbol genealógico, pero contiene múltiples errores sintácticos y lógicos.



FIN