

## **PROYECTO FINAL - CS**

### **SISTEMA DE ELECCIONES**

**AVENDAÑO RODRIGUEZ SEBASTIAN - 20232020101**  
**MENDIVELSO MARTINEZ SERGIO NICOLAS - 20231020227**

**DICIEMBRE 5, 2025**

# ÍNDICE

<b>1. DESCRIPCIÓN</b>	<b>3</b>
1.1. RESUMEN . . . . .	3
1.2. ¿POR QUÉ ES IMPORTANTE? . . . . .	3
1.3. INCLUYE POR DEFECTO: . . . . .	3
<b>2. PLAN DE DESARROLLO</b>	<b>4</b>
2.1. PROBLEMÁTICA ORIGINAL . . . . .	4
2.2. REQUISITOS IDENTIFICADOS . . . . .	4
2.2.1. REQUISITOS FUNCIONALES . . . . .	4
2.2.2. REQUISITOS NO-FUNCIONALES . . . . .	5
<b>3. OBJETIVOS DEL PROYECTO</b>	<b>6</b>
3.1. OBJETIVO GENERAL . . . . .	6
3.2. OBJETIVOS ESPECÍFICOS . . . . .	6
<b>4. DISEÑO DE LA SOLUCIÓN</b>	<b>7</b>
4.1. VISIÓN GENERAL . . . . .	7
4.2. PRINCIPIOS DE DISEÑO APLICADOS . . . . .	7
4.2.1. SEPARACIÓN DE RESPONSABILIDADES . . . . .	7
4.2.2. COHESIÓN ALTA . . . . .	7
4.2.3. ACOPLAMIENTO BAJO . . . . .	7
4.2.4. ESCALABILIDAD . . . . .	8
<b>5. ESTRUCTURAS DE DATOS</b>	<b>9</b>
5.1. GENERACIÓN DEL ESQUELETO DE STRUCTS . . . . .	9
5.2. CÓDIGO HEXADECIMAL PARA IDENTIFICACIÓN . . . . .	9
5.3. DIVISIÓN DEL SISTEMA EN LISTAS ESPECIALIZADAS . . . . .	9
5.4. CONEXIÓN JERÁRQUICA SIMILAR A UN ÁRBOL . . . . .	9
5.5. ORDENAMIENTO AUTOMÁTICO DE LISTAS . . . . .	9
5.6. EJEMPLO DE LAS RELACIONES . . . . .	10
5.7. CLASES ENUM . . . . .	10
5.8. ESTRUCTURA PAÍS . . . . .	10
5.8.1. RELACIONES . . . . .	10
5.9. ESTRUCTURA REGIÓN . . . . .	11
5.9.1. RELACIONES . . . . .	11
5.9.2. CÁLCULO DE CENSO . . . . .	11
5.10. ESTRUCTURA: CIUDAD (CIUDAD) . . . . .	11
5.11. ESTRUCTURA: PARTIDO POLÍTICO (PARTIDO) . . . . .	11
5.12. ESTRUCTURA: CANDIDATO (CANDIDATO) . . . . .	11
5.12.1. VALIDACIONES . . . . .	12
<b>6. MÓDULOS Y COMPONENTES</b>	<b>13</b>
6.1. MÓDULO: CARGAR DATOS . . . . .	13
6.1.1. MÉTODOS PRINCIPALES . . . . .	13
6.1.2. CÓMO SE LEEN LOS DATOS . . . . .	13
6.1.3. CÓMO SE SEPARAN LOS DATOS . . . . .	13
6.2. MÓDULO: DATOS ELECTORAL . . . . .	13
6.2.1. CREAR ELEMENTOS NUEVOS . . . . .	13
6.2.2. CONSULTAR INFORMACIÓN . . . . .	14
6.3. MÓDULO: UTILIDADES ANALISIS . . . . .	14
6.3.1. FUNCIONES PRINCIPALES . . . . .	14
6.4. MÓDULO: SISTEMA ELECTORAL . . . . .	14
6.5. MÓDULO: SIMULACIÓN ELECTORAL . . . . .	14

6.5.1. RESULTADOS . . . . .	15
7. ESTRUCTURAS DE DATOS EN MEMORIA SECUNDARIA	16
7.1. ARCHIVOS PLANOS . . . . .	16
7.1.1. PARTIDOS.TXT . . . . .	16
7.1.2. REGIONES.TXT . . . . .	16
7.1.3. CIUDADES.TXT . . . . .	16
7.1.4. CANDIDATOS.TXT . . . . .	16
7.2. DIVISIÓN DEL SISTEMA EN LISTAS . . . . .	16
7.2.1. USO EN DATOSELECTORAL . . . . .	16
7.2.2. RELACIONES ENTRE DATOS . . . . .	17
7.2.3. VENTAJAS DE USAR LISTAS . . . . .	17
7.2.4. EJEMPLO DE USO . . . . .	17
7.2.5. ¿POR QUÉ LISTAS Y NO ÁRBOLES? . . . . .	18
8. INTERFAZ DE USUARIO	19
8.1. MENÚ PRINCIPAL . . . . .	19
8.2. MENÚ DE CONSULTAS . . . . .	19
9. CONCLUSIONES	20
9.1. OBJETIVOS LOGRADOS . . . . .	20

# 1. DESCRIPCIÓN

Este proyecto es expuesto como proyecto final dentro del marco de notas de la asignatura de Ciencias de la Computacion 1. Donde se evidencia la siguiente problemática:

## 1.1. Resumen

La Registraduría Nacional del Estado Civil de un país del sagrado corazón requiere un software robusto que permita:

1. **Generar aleatoriamente tarjetones electorales** para alcaldías locales y presidencia
2. **Simular el proceso electoral completo** con validaciones
3. **Reportar resultados** mediante estadísticas y reportes
4. **Validar la calidad del sistema** antes de su implementación nacional

## 1.2. ¿Por qué es importante?

- Automatiza la generación de tarjetones respetando la estructura política
- Se simula comportamientos electorales reales con restricciones válidas
- Genera reportes estadísticos para su debida verificación
- Permite múltiples simulaciones para validación y calibración

## 1.3. Incluye por defecto:

- Gestión de 10 partidos políticos
- Elecciones en 8 capitales de departamentos
- 5 candidatos presidenciales con fórmulas vicepresidenciales
- 4 candidatos por alcaldía en cada capital
- Generación aleatoria de votos con restricciones
- Reportes de estadísticas por región y nacional
- Detección de segunda vuelta presidencial

## 2. PLAN DE DESARROLLO

### 2.1. Problemática Original

*“La Registraduría Nacional del Estado Civil quiere hacer en un mismo día las elecciones de alcaldes y de presidente para ahorrar gastos. Para esto necesita un programa que pueda crear tarjetones al azar para cada ciudad y para la presidencia (incluyendo voto en blanco), simular todo el proceso de votación y mostrar los resultados y algunas estadísticas.”*

### 2.2. Requisitos Identificados

#### 2.2.1. Requisitos Funcionales

##### Consultas sobre candidatos

- Ver candidatos por partido y región
- Ver candidatos por partido en todas las capitales
- Ver candidatos por ciudad y partido
- Ver candidatos presidenciales con sus vicepresidentes

##### Visualización de datos

- Ver tarjetones de alcaldía
- Ver tarjetones de presidencia
- Ver el número de personas que pueden votar por ciudad

##### Gestión de datos pre-simulación

- Agregar candidatos
- Eliminar candidatos
- Editar información de candidatos

##### Simulación electoral

- Generar votos al azar
- No sobre pasar el número máximo de votantes por ciudad
- Calcular automáticamente la edad de los candidatos
- Encontrar alcaldes ganadores
- Encontrar el candidato a presidencia ganador
- Detectar si se necesita segunda vuelta

##### Estadísticas

- Mostrar resultados por región para las alcaldías
- Mostrar resultados nacionales para la presidencia
- Mostrar datos separados por sexo
- Mostrar datos separados por partido

**2.2.2. Requisitos No-Funcionales**

- Código claro y fácil de entender
- Buen manejo de los datos
- Respuesta rápida del programa
- Estructuras de datos
- Revisiones internas para evitar errores

### **3. OBJETIVOS DEL PROYECTO**

#### **3.1. Objetivo General**

Crear un sistema que simule las elecciones de presidente y de alcaldes en las capitales de los departamentos. El sistema debe generar tarjetones al azar, procesar los votos y mostrar resultados y estadísticas claras y confiables.

#### **3.2. Objetivos Específicos**

- Usar estructuras de datos para manejar la información electoral
- Hacer búsquedas rápidas y eficientes dentro del sistema
- Crear tarjetones al azar para cada capital
- Simular votos
- Calcular los resultados de las elecciones
- Mostrar estadísticas por región, partido y género
- Detectar casos especiales como segunda vuelta

## 4. DISEÑO DE LA SOLUCIÓN

### 4.1. Visión General

El sistema está organizado de tal manera que cada parte del programa tiene una tarea clara y no depende demasiado de las otras.

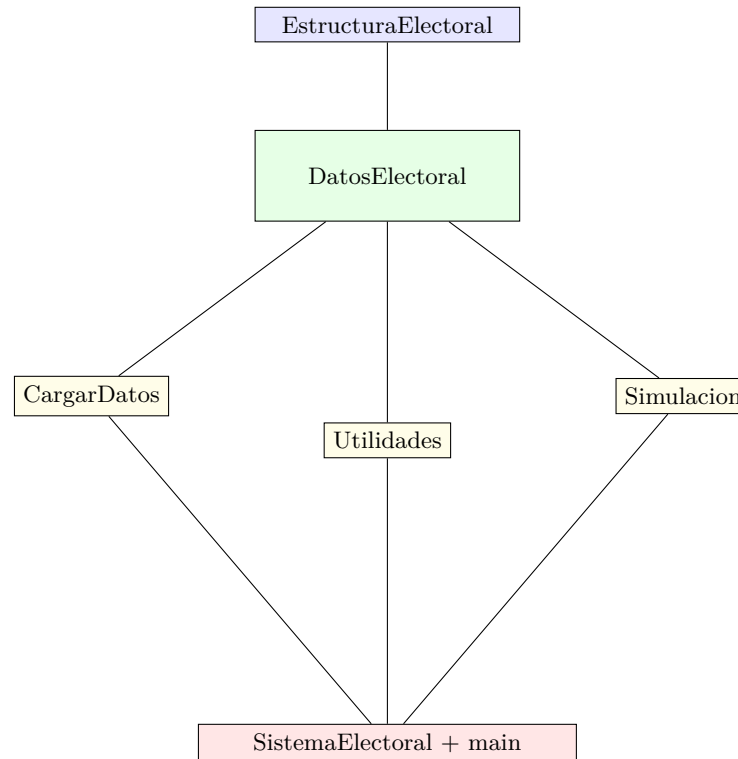


Figura 1: Relaciones simples entre las partes del sistema y como está organizado

### 4.2. Principios de Diseño Aplicados

#### 4.2.1. Separación de Responsabilidades

- Cada parte del sistema tiene una tarea clara
- Las partes se comunican de manera sencilla
- Se evita que los módulos dependan demasiado entre sí

#### 4.2.2. Cohesión Alta

- Las funciones parecidas están en la misma clase
- Los datos importantes están protegidos
- Se usan funciones generales cuando se necesita algo para todo el sistema

#### 4.2.3. Acoplamiento Bajo

- Se usan referencias simples para conectar módulos
- Se evita cargar archivos o dependencias que no se necesitan

- Se anuncian clases sin detalles cuando es posible para no sobrecargar el sistema

#### **4.2.4. Escalabilidad**

- Es fácil agregar nuevas consultas o funciones
- El sistema puede crecer sin tener que volver a hacerlo
- Los datos están separados de la lógica

## 5. ESTRUCTURAS DE DATOS

### 5.1. Generación del Esqueleto de Structs

Se construyó un esqueleto de estructuras (*structs*) que modela las entidades del sistema electoral: **País**, **Región**, **Ciudad**, **Partido** y **Candidato**. Cada una contiene únicamente la información esencial para mantener el sistema flexible, modular y escalable.

### 5.2. Código Hexadecimal para Identificación

Cada entidad posee un campo `codigo`, el cual puede almacenar un valor generado a partir del **hexadecimal del nombre u otra variable única**. El propósito es:

- permitir un ordenamiento homogéneo y eficiente,
- clasificar entidades usando un tipo de dato común,
- facilitar búsquedas, filtrados y acceso rápido.

Este identificador hexadecimal será utilizado para mantener las listas ordenadas automáticamente.

### 5.3. División del Sistema en Listas Especializadas

Cada entidad del sistema se almacena en una lista completamente separada:

- Lista de Países
- Lista de Regiones
- Lista de Ciudades
- Lista de Partidos
- Lista de Candidatos

### 5.4. Conexión Jerárquica Similar a un Árbol

Aunque los datos se almacenan en listas, la relación entre estructuras respeta una jerarquía:

- Un **País** contiene múltiples candidatos presidenciales.
- Una **Región** pertenece a un País (*muchas a uno*).
- Una **Ciudad** pertenece a una Región (*muchas a uno*).
- Los **Candidatos** no poseen una relación directa entre sí más allá del Partido.

Esto crea un modelo de datos que funciona como un árbol conceptual:

País → Región → Ciudad

### 5.5. Ordenamiento Automático de Listas

Cada lista ordenará sus elementos inmediatamente después de insertarlos, basándose en el código hexadecimal, lo que permite consultas más rápidas y una organización constante.

## 5.6. Ejemplo de las relaciones

- El **País** contiene una `Lista<Candidato*>` con candidatos a la presidencia.
- Cada **Región** tiene un puntero a su País padre:

Región → País

- Cada **Ciudad** tiene un puntero a su Región padre:

Ciudad → Región

- Los candidatos se enlazan únicamente con su Partido y, si va al caso, con su fórmula.

Este diseño permite representar de manera eficiente el modelo electoral, respetando la jerarquía institucional sin sacrificar velocidad de acceso ni facilidad para mostrar información general.

A continuación, se presenta la estructura principal de los objetos en código.

## 5.7. Clases Enum

```

1  enum class Sexo {
2      Masculino,
3      Femenino
4  };
5
6  enum class EstadoCivil {
7      Soltero,
8      Casado,
9      Divorciado,
10     UnionLibre
11 };
12
13 enum class TipoCandidato {
14     ALCALDE,
15     PRESIDENTE,
16     VICEPRESIDENTE
17 };

```

## 5.8. Estructura País

```

1  struct Pais {
2      std::string codigo;           / Código único del país
3      std::string nombre;          / Ejemplo: "Colombia"
4      std::vector<Candidato*> candidatosPresidencia;
5      std::vector<Region*> regiones;
6  };

```

### 5.8.1. Relaciones

- Contiene varias regiones
- Contiene los candidatos a la presidencia

## 5.9. Estructura Región

```
1 struct Region {
2     std::string codigo;           / Identificador
3     std::string nombre;          / Ejemplo: "Andina Central"
4     std::vector<Ciudad*> ciudades;
5     int censoElectoral = 0;
6     Pais* pais = nullptr;
7 };
```

### 5.9.1. Relaciones

- Pertenece a un país
- Contiene varias ciudades

### 5.9.2. Cálculo de Censo

Cada ciudad cuenta con su Censo correspondiente. El censo de la región es la suma de los censos de todas las ciudades.

## 5.10. Estructura: Ciudad (Ciudad)

```
1 struct Ciudad {
2     std::string codigo;
3     std::string nombre;
4     Region* region = nullptr;
5     std::vector<Candidato*> candidatosAlcaldia;
6     int censoElectoral = 0;
7 };
8
```

## 5.11. Estructura: Partido Político (Partido)

```
1 struct Partido {
2     std::string codigo;
3     std::string nombre;
4     std::string representanteLegal;
5     bool legal = false;
6 };
```

## 5.12. Estructura: Candidato (Candidato)

```
1 struct Candidato {
2     / Información personal
3     std::string nombre;
4     std::string apellido;
5     std::string identificacion;
6     std::tm fechaNacimiento;
7     Sexo sexo;
8     EstadoCivil estadoCivil;
9
10    / Información de ciudades
11    Ciudad* ciudadNacimiento = nullptr;
```

```
12 Ciudad* ciudadResidencia = nullptr;
13
14 / Información electoral
15 Partido* partido = nullptr;
16 TipoCandidato tipo;
17 Ciudad* ciudadAspirante = nullptr;
18 Candidato* vicepresidente = nullptr;
19
20 bool esValido() const;
21 };
```

#### 5.12.1. Validaciones

- Nombre, apellido e identificación no pueden estar vacíos
- Debe tener un partido asignado
- Su edad debe estar entre 18 y 100 años
- Si es candidato a alcalde, debe vivir en la ciudad donde se postula

## 6. MÓDULOS Y COMPONENTES

### 6.1. Módulo: CargarDatos

Archivos: CargarDatos.h y CargarDatos.cpp

Responsabilidad: Leer archivos de texto y llenar las estructuras en memoria.

#### 6.1.1. Métodos principales

```
1 static bool cargarTodosLosDatos(  
2     DatosElectoral& sistema,  
3     const std::string& rutaPartidos,  
4     const std::string& rutaRegiones,  
5     const std::string& rutaCiudades,  
6     const std::string& rutaCandidatos,  
7     Pais* pais  
8 );  
9  
10 static bool cargarPartidos(DatosElectoral& sistema,  
11                             const std::string& ruta);  
12 static bool cargarRegiones(DatosElectoral& sistema,  
13                             const std::string& ruta, Pais* pais);  
14 static bool cargarCiudades(DatosElectoral& sistema,  
15                             const std::string& ruta);  
16 static bool cargarCandidatos(DatosElectoral& sistema,  
17                             const std::string& ruta, Pais* pais);
```

#### 6.1.2. Cómo se leen los datos

- Abrir el archivo con `ifstream`
- Leer una línea por vez
- Revisar que el formato sea correcto y que no falte nada

#### 6.1.3. Cómo se separan los datos

- Dividir cada línea usando la coma como separador
- Convertir los datos al tipo necesario (texto a número, fecha, etc.)
- Quitar espacios extra con `trim()`

### 6.2. Módulo: DatosElectoral

Archivo: DatosElectoral.h y DatosElectoral.cpp

Responsabilidad: Manejar todos los datos del sistema.

#### 6.2.1. Crear elementos nuevos

```
1 Pais* crearPais(std::string nombre, ...);  
2 Region* crearRegion(std::string nombre, ...);  
3 Ciudad* crearCiudad(std::string nombre, ...);  
4 Partido* crearPartido(std::string nombre, ...);
```

### 6.2.2. Consultar información

```
1 std::vector<Ciudad*> obtenerCiudadesElectorales();
2 std::vector<Partido*> obtenerPartidosLegales();
3 std::vector<Region*> obtenerListaRegiones();
4
5 std::vector<pair<Candidato*, Candidato*>>
6   candidatosPresidenciales();
```

## 6.3. Módulo: UtilidadesAnálisis

**Archivo:** UtilidadesAnálisis.h y UtilidadesAnálisis.cpp

**Responsabilidad:** Funciones de apoyo para leer, revisar datos y mostrar información.

### 6.3.1. Funciones principales

```
1 / Leer datos
2 static std::tm parsearFecha(const std::string& fecha);
3 static Sexo parsearSexo(char opcion);
4 static EstadoCivil parsearEstadoCivil(int opcion);
5
6 / Cálculos sencillos
7 static int calcularEdad(const std::tm& fechaNacimiento);
8 static unsigned long long hashToULL(const std::string& input);
9
10 / Impresión de información
11 static void imprimirTarjetonAlcaldia(Ciudad* ciudad);
12 static void imprimirTarjetonPresidencia(Pais* pais);
```

## 6.4. Módulo: SistemaElectoral

**Archivo:** SistemaElectoral.h y SistemaElectoral.cpp

**Responsabilidad:** Menús e interacción con la persona usuaria.

```
1 namespace SistemaElectoral {
2     Ciudad* seleccionarCiudad(DatosElectoral& sistema);
3     Partido* seleccionarPartido(DatosElectoral& sistema);
4
5     void agregarCandidato(DatosElectoral& sistema, Pais* pais);
6     void eliminarCandidato(DatosElectoral& sistema);
7     void modificarCandidato(DatosElectoral& sistema);
8
9     void menuGestionCandidatos(DatosElectoral& sistema,
10                                Pais* pais);
11     void menuConsultas(DatosElectoral& sistema, Pais* pais);
12 }
```

## 6.5. Módulo: SimulacionElectoral

**Archivo:** SimulacionElectoral.h y SimulacionElectoral.cpp

**Responsabilidad:** Manejar la simulación de las votaciones .

### 6.5.1. Resultados

```
1 struct ResultadosCiudad {
2     Ciudad* ciudad;
3     std::<Candidato*, int> votosAlcaldia;
4     int votosEnBlancoAlcaldia = 0;
5     int votosNulosAlcaldia = 0;
6     int abstencionAlcaldia = 0;
7     Candidato* ganadorAlcaldia = nullptr;
8 };
9
10 struct ResultadosNacionales {
11     std::map<Candidato*, int> votosPresidencia;
12     int votosEnBlancoPresidencia = 0;
13     int votosNulosPresidencia = 0;
14     int abstencionPresidencia = 0;
15     Candidato* ganadorPresidencia = nullptr;
16     bool requiereSegundaVuelta = false;
17 };
```

## 7. ESTRUCTURAS DE DATOS EN MEMORIA SECUNDARIA

### 7.1. Archivos Planos

#### 7.1.1. PARTIDOS.txt

**Formato:** nombre,representanteLegal,legal

```
1 Centro Democrático,Juan Perez,true
2 Partido Liberal,Carlos Garcia,true
3 Polo Democratico,Maria Lopez,true
4 Cambio Radical,Roberto Silva,true
5 Alianza Verde,Ana Martinez,true
```

#### 7.1.2. REGIONES.txt

**Formato:** nombreRegion:ciudad1,ciudad2,ciudad3

```
1 Andina Central:Bogotá,Medellín,Cali
2 Caribe:Cartagena,Santa Marta,Barranquilla
3 Pacifica:Buenaventura,Quibdó
```

#### 7.1.3. CIUDADES.txt

**Formato:** nombreCiudad,nombreRegion,censoElectoral

```
1 Bogotá,Andina Central,2500000
2 Medellín,Andina Central,1800000
3 Cali,Andina Central,1600000
```

#### 7.1.4. CANDIDATOS.txt

**Formato:** nombre,apellido,ID,sexo,estadoCivil,fechaNac,ciudadNac,ciudadRes,partido,tipo[,ciudadAspirante]

```
1 Carlos,Gomez,1234567890,M,Casado,1975-03-15,Bogotá,Bogotá,Centro Democrático,Alcalde
   ,Bogotá
2 Ana,Rodriguez,0987654321,F,Soltero,1980-08-22,Medellín,Medellín,Partido Liberal,
   Alcalde,Medellín
```

### 7.2. División del Sistema en Listas

#### 7.2.1. Uso en DatosElectoral

```
1 / Listas separadas para cada tipo
2 class DatosElectoral {
3 private:
4     std::vector<Pais*> paises;
5     std::vector<Region*> regiones;
6     std::vector<Ciudad*> ciudades;
7     std::vector<Candidato*> candidatosAlcaldia;
8     std::vector<Candidato*> candidatosPresidenciaLista;
9     std::vector<Partido*> partidos;
10
11 public:
12     std::vector<Pais*>& obtenerListaPaises();
13     std::vector<Region*>& obtenerListaRegiones();
14     std::vector<Ciudad*>& obtenerListaCiudades();
```

```
15     std::vector<Candidato*>& obtenerListaCandidatos();
16     std::vector<Partido*>& obtenerListaPartidos();
17 };
```

### 7.2.2. Relaciones entre Datos

Aunque cada cosa está en su propia lista, siguen conectadas mediante punteros:

```
1  / Una Ciudad pertenece a una Región
2  struct Ciudad {
3      std::string nombre;
4      Region* region;
5      std::vector<Candidato*> candidatosAlcaldia;
6      int censoElectoral;
7  };
8
9  / Un Candidato pertenece a un Partido y Ciudades
10 struct Candidato {
11     std::string nombre;
12     Partido* partido;
13     Ciudad* ciudadNacimiento;
14     Ciudad* ciudadResidencia;
15     Candidato* vicepresidente;
16 };
17
18 / Una Región pertenece a un País
19 struct Region {
20     std::string nombre;
21     Pais* pais;
22     std::vector<Ciudad*> ciudades;
23 };
```

### 7.2.3. Ventajas de Usar Listas

- Son independientes y no duplican datos
- Se pueden recorrer fácilmente
- Es sencillo agregar o quitar elementos
- Cada lista tiene una función clara
- Se pueden añadir nuevos tipos sin complicaciones
- Es fácil obtener datos generales

### 7.2.4. Ejemplo de Uso

```
1  / Crear toda la estructura
2  DatosElectoral sistema;
3
4  / 1. Crear un País
5  Pais* pais = new Pais();
6  pais->nombre = "Colombia";
7  sistema.obtenerListaPaises().push_back(pais);
8
9  / 2. Crear una Región
10 Region* region = new Region();
```

```
11 region->nombre = "Andina Central";
12 region->pais = pais;
13 sistema.obtenerListaRegiones().push_back(region);
14
15 / 3. Crear una Ciudad
16 Ciudad* ciudad = new Ciudad();
17 ciudad->nombre = "Bogotá";
18 ciudad->region = region;
19 sistema.obtenerListaCiudades().push_back(ciudad);
20
21 / 4. Crear un Partido
22 Partido* partido = new Partido();
23 partido->nombre = "Centro Democrático";
24 sistema.obtenerListaPartidos().push_back(partido);
25
26 / 5. Crear un Candidato
27 Candidato* candidato = new Candidato();
28 candidato->nombre = "Carlos";
29 candidato->partido = partido;
30 candidato->ciudadResidencia = ciudad;
31 sistema.obtenerListaCandidatos().push_back(candidato);
```

#### 7.2.5. ¿Por Qué Listas y No Árboles?

- Es fácil recorrer todos los elementos
- Las estadísticas se consiguen rápidamente
- La estructura es simple y clara
- No obliga a tener jerarquías estrictas
- Agregar elementos es muy rápido
- No hay dependencias complicadas

Las relaciones importantes se manejan por punteros, no por estructuras rígidas como árboles.

## 8. INTERFAZ DE USUARIO

### 8.1. Menú Principal

```
1 +-----+
2 | SISTEMA ELECTORAL - MENU PRINCIPAL |
3 |-----|
4 | 1. Consultas |
5 | 2. Gestión de candidatos |
6 | 3. Simulación de elecciones |
7 | 0. Salir |
8 +-----+
9
10 Seleccione una opción:
```

### 8.2. Menú de Consultas

```
1 +-----+
2 | MENU DE CONSULTAS |
3 |-----|
4 | 1. Ciudades electorales |
5 | 2. Partidos legales |
6 | 3. Candidatos por ciudad |
7 | 4. Candidatos presidenciales |
8 | 5. Tarjetón de alcaldía |
9 | 6. Tarjetón de presidencia |
10 | 7. Censo electoral |
11 | 0. Volver |
12 +-----+
13
14 Seleccione una opción:
```

## 9. CONCLUSIONES

### 9.1. Objetivos logrados

El desarrollo de este sistema electoral nos ayudo a entender de forma práctica conceptos principales que hemos estado viendo en la asignatura de Ciencias de la Computación; cosas como el diseño de estructuras de datos, análisis de algoritmos y programación modular en C++, permitieron nuestro mejor entendimiento de las cosas. Se modelaron entidades como país, región, ciudad, partido y candidato usando structs y vectores, por lo que mantuvimos relaciones mediante punteros para lograr una organización clara y eficiente.

En términos algorítmicos, se implementaron búsquedas optimizadas y se controló que operaciones críticas respetaran invariantes como que la suma de votos no supere el censo electoral. Esto con el fin de aplicar conceptos sobre complejidad temporal y corrección, incluso en proyectos de simulación.

La arquitectura por módulos (carga de datos, gestor central, utilidades, sistema de menús y simulación) favorece la cohesión y reduce el acoplamiento, lo que facilita mantener y extender el código en caso que se necesite.