



LAUREATE INTERNATIONAL UNIVERSITIES®

LÓGICA Y
PROGRAMACIÓN
ESTRUCTURADA

Profr. Luis Guzman Gonzaga

**Unidad 5.
Archivos**

**Actividad 8.
Proyecto Integrador
Etapa 3**

Nailea Mitchel Falcon Triana

940066032

La Paz, Baja California Sur, 6 de Diciembre de 2025

UNIVERSIDAD DEL VALLE DE MÉXICO

SISTEMA DE GESTIÓN

INTRODUCCIÓN:

Se ha diseñado una solución de software que consiste en un Catálogo Botánico Inteligente, una aplicación de consola desarrollada íntegramente en C++. Esta herramienta está diseñada no solo para el simple almacenamiento de datos, sino para su gestión de una manera significativa, estructurada y, fundamentalmente, escalable.

La elección de este tema no es arbitraria. Si bien un catálogo de plantas podría parecer, a primera vista, un inventario simple, la propuesta innovadora de este proyecto radica en su arquitectura de datos. Se ha priorizado la profundidad y la utilidad de la información sobre la mera cantidad.

En lugar de limitarse a campos básicos como el nombre común, la especie y una descripción textual, el núcleo de este sistema emplea estructuras anidadas en C++. Esta decisión de diseño permite registrar un conjunto detallado de parámetros de cuidado vitales y cuantificables para cada espécimen botánico.

Estos parámetros incluyen, de forma específica, los rangos óptimos de humedad del suelo, los umbrales mínimos y máximos de temperatura ambiente, y el fotoperiodo exacto que cada planta requiere para prosperar. Esta granularidad en los datos es lo que define el carácter inteligente del catálogo.

Este enfoque de diseño transforma la aplicación de un simple contenedor de información pasiva a una base de conocimiento activa y funcional. El sistema no solo registra qué es la planta, sino que comprende cómo debe ser cuidada, sentando así las bases para una futura gestión proactiva y automatizada. Paralelamente, se han establecido las bases del código fuente, implementando una versión funcional del menú principal.

Este planteamiento no solo cumple con los objetivos de la Etapa 1, sino que establece una plataforma sólida y bien estructurada para las futuras etapas del proyecto. Las operaciones de baja, modificación y búsqueda, que se implementarán posteriormente, operarán sobre este conjunto de datos enriquecido. Esto permitirá, en fases futuras, desarrollar el propósito final de la aplicación: un sistema de asistencia inteligente capaz de generar recordatorios, alertas de cuidado y diagnósticos básicos basados en los parámetros específicos de cada planta registrada.

ESTRUCTURA CATÁLOGO

DESARROLLO:

1. DEFINICIÓN DEL CATÁLOGO Y DISEÑO DE INTERFAZ:

Esta sección detalla el concepto del Catálogo Botánico Inteligente, define su alcance funcional y presenta el diseño de las pantallas de consola de texto que facilitan su gestión.

2. DESCRIPCIÓN Y ALCANCE DEL CATÁLOGO:

El Catálogo Botánico Inteligente es un sistema de gestión de perfiles de plantas diseñado para almacenar, organizar y consultar los parámetros de cuidado específicos de cada espécimen. El sistema está concebido como la base de datos central para un futuro asistente de jardinería personal o un sistema de monitoreo automatizado.

La entidad principal del catálogo es la Planta. A diferencia de un catálogo tradicional, cada registro de Planta contendrá una subestructura anidada denominada **ParametrosCuidado**. Esta estructura es la característica fundamental del proyecto, ya que almacena los rangos operativos (mínimos y máximos) que la planta tolera para prosperar.

3. OPERACIONES DEL CATÁLOGO:

- **Altas:** Registrar una nueva planta en la colección. El usuario deberá ingresar no solo el nombre (ej. Helecho de Boston), sino también sus parámetros ideales (ej. Humedad 50%-70%, Temp 18°C-24°C).
- **Bajas:** Eliminar un registro de planta del catálogo, por ejemplo, si la planta es descartada de la colección.
- **Modificaciones:** Actualizar la información de una planta. Esta función es crucial si el usuario aprende más sobre el cuidado de una planta específica y necesita ajustar sus rangos de parámetros.
- **Consultas:** Permitir al usuario ver la lista completa de plantas o buscar especímenes que cumplan ciertos criterios (ej. Mostrar todas las plantas que toleran temperaturas por debajo de 15°C).

DISEÑO DE INTERFAZ

PANTALLAS DE CONSOLA:

PÁGINA PRINCIPAL

Dado que la aplicación se basa en C++, la interfaz de usuario será textual, limpia y basada en menús. Se priorizará la navegación intuitiva.

Conforme a las instrucciones [1], estos diseños son prototipos visuales y no tienen funcionalidad en esta etapa.

Pantalla 1: Menú Principal (Sigue el ejemplo base de "Menú de opciones" [1])

```
+-----+
| CATÁLOGO BOTÁNICO INTELIGENTE |
+-----+
| |
| Menú de Opciones: |
| |
| 1. Alta de Nueva Planta |
| 2. Baja de Planta |
| 3. Modificación de Planta |
| 4. Consulta de Catálogo |
| 5. Salir |
| |
+-----+
| Seleccione una opción entre [1...5]: _ |
+-----+
```

```
+-----+
| 1. ALTA DE NUEVA PLANTA |
+-----+
| |
| Ingrese el ID de la planta (ej. 101): _ |
| Ingrese el Nombre Común: _ |
| Ingrese la Especie (Científico): _ |
| |
| --- Parámetros de Cuidado Óptimo --- |
| |
| Humedad del Suelo (MIN %): _ |
| Humedad del Suelo (MAX %): _ |
| |
| Temperatura Amb. (MIN C): _ |
| Temperatura Amb. (MAX C): _ |
| |
| Horas de Luz Directa (MIN): _ |
| Horas de Luz Directa (MAX): _ |
| |
+-----+
| [Mensaje: Planta guardada con éxito.] |
+-----+
```

ALTA NUEVA PLANTA

Dado que la aplicación se basa en C++, la interfaz de usuario será textual, limpia y basada en menús. Se priorizará la navegación intuitiva.

Idea Visual 2: Pantalla de "Alta de Planta" La interfaz guía al usuario campo por campo, solicitando los parámetros de cuidado de forma estructurada.

DISEÑO DE INTERFAZ

PANTALLAS DE CONSOLA:

```
+-----+
| 2. BAJA DE PLANTA |
+-----+
|
| Ingrese el ID de la planta a eliminar: _ |
|
| --- Verificación --- |
| Planta: Helecho de Boston (ID: 101) |
| Especie: Nephrolepis exaltata |
|
| ¿Está seguro que desea eliminar? (S/N): _ |
|
+-----+
|
| [Mensaje: Planta eliminada con éxito.] |
+-----+
```

PÁGINA PRINCIPAL

Dado que la aplicación se basa en C++, la interfaz de usuario será textual, limpia y basada en menús. Se priorizará la navegación intuitiva.

Pantalla 3: Baja de Planta (Flujo para la operación de Bajas [1])

CONSULTA DE CATÁLOGO

Dado que la aplicación se basa en C++, la interfaz de usuario será textual, limpia y basada en menús. Se priorizará la navegación intuitiva.

Idea Visual 4: Pantalla de "Consultar Catálogo" La consulta muestra los datos de forma tabulada y limpia para fácil lectura.

```
+-----+
| 4. CONSULTA DE CATÁLOGO |
+-----+
|
| --- Plantas Registradas (3) --- |
|
| Helecho de Boston |
| -> Humedad: 50-70% | Temp: 18-24C |
|
| Suculenta (Echeveria) |
| -> Humedad: 10-20% | Temp: 20-28C |
|
| Orquídea (Phalaenopsis) |
| -> Humedad: 60-80% | Temp: 22-27C |
|
+-----+
|
| [Presione Enter para volver al Menú...] |
+-----+
```

OPERACIÓN DE CONCEPTOS

ESTRUCTURA DE PROGRAMACIÓN:

1. OPERACIÓN DE CONCEPTOS UTILIZANDO LENGUAJE DE PROGRAMACIÓN

Esta sección presenta el Planteamiento de los conceptos del catálogo utilizando el editor o IDE recomendado, que en este caso es C++.

El pilar de la programación estructurada para este proyecto es el uso de struct (estructuras) para definir nuestro modelo de datos. Para lograr el concepto novedoso, no basta con una sola estructura; necesitamos anidarlas.

Definimos dos estructuras principales:

- 1. struct ParametrosCuidado:** Una estructura interna que agrupa todos los rangos numéricos que definen el cuidado de la planta.
- 2. struct Planta:** La estructura principal del catálogo, que contiene la información básica (ID, nombre) y, crucialmente, contiene una variable de tipo ParametrosCuidado en su interior.

El código fuente se organizaría de la siguiente manera:

Archivo: [ParametrosCuidado.h \(Definición de la estructura anidada\)](#)

```
C++  
  
/*  
 * ParametrosCuidado.h  
 * Define la estructura para los rangos de cuidado óptimo.  
 */  
#pragma once // Evita la inclusión múltiple  
  
struct ParametrosCuidado {  
    // Rangos de humedad del suelo (porcentaje)  
    int humedad_min;  
    int humedad_max;  
  
    // Rangos de temperatura ambiente (Celsius)  
    int temp_min;  
    int temp_max;  
  
    // Rango de horas de luz directa por día  
    int luz_min;  
    int luz_max;  
};
```

OPERACIÓN DE CONCEPTOS

ESTRUCTURA DE PROGRAMACIÓN:

Archivo: `Planta.h` (Definición de la estructura principal)

C++

```
/*
 * Planta.h
 * Define la estructura principal del catálogo,
 * que anida los parámetros de cuidado.
 */
#pragma once
#include "ParametrosCuidado.h" // Incluye la definición de la otra estructura
#include <string>

struct Planta {
    int id_planta;
    std::string nombre_comun;
    std::string especie_cientifica;

    // Anidación de la estructura de parámetros
    ParametrosCuidado cuidados_optimos;
};
```

Archivo: `main.cpp` (Programa principal de prueba Etapa 1) Este archivo demuestra que los conceptos compilan y se puede crear una instancia de la estructura `Planta`. Esto cumple con el requisito de compilar el programa y proporcionar evidencia de ejecución.[1,1]

C++

```
/*
 * main.cpp
 * Proyecto Integrador Etapa 1: Catálogo Botánico Inteligente
 *
 * Propósito: Demostrar que las estructuras de datos (conceptos)
 * del catálogo compilan correctamente y están enlazadas.
 *
 * (La funcionalidad de Alta, Baja, etc., se implementará
 * en las Etapas 2 y 3).
 */

#include <iostream>
#include "Planta.h" // Incluye nuestra estructura principal

int main() {

    // 1. Declarar una variable de tipo 'Planta'
    // Si esto compila, las estructuras anidadas (Planta y
    // ParametrosCuidado) están correctamente definidas.
    Planta planta_de_prueba;

    // Asignar datos de prueba
    planta_de_prueba.id_planta = 101;
    planta_de_prueba.nombre_comun = "Melecho de Boston";

    // Asignar datos a la estructura ANIDADA
    planta_de_prueba.cuidados_optimos.humedad_min = 50;
    planta_de_prueba.cuidados_optimos.humedad_max = 70;
    planta_de_prueba.cuidados_optimos.temp_min = 18;

    // 2. Imprimir mensaje de éxito en consola (Evidencia de ejecución)
    std::cout << "+-----+" << std::endl;
    std::cout << "| PROYECTO INTEGRADOR ETAPA 1 |" << std::endl;
    std::cout << "| CATÁLOGO BOTÁNICO INTELIGENTE |" << std::endl;
    std::cout << "| |" << std::endl;
    std::cout << "| | -> Programa compilado con éxito. |" << std::endl;
    std::cout << "| | -> Estructuras 'Planta' y |" << std::endl;
    std::cout << "| | 'ParametrosCuidado' enlazadas. |" << std::endl;
    std::cout << "+-----+" << std::endl;

    return 0;
}
```

OPERACIÓN DE CONCEPTOS

SISTEMA DE GESTIÓN VISUAL:



Simbología

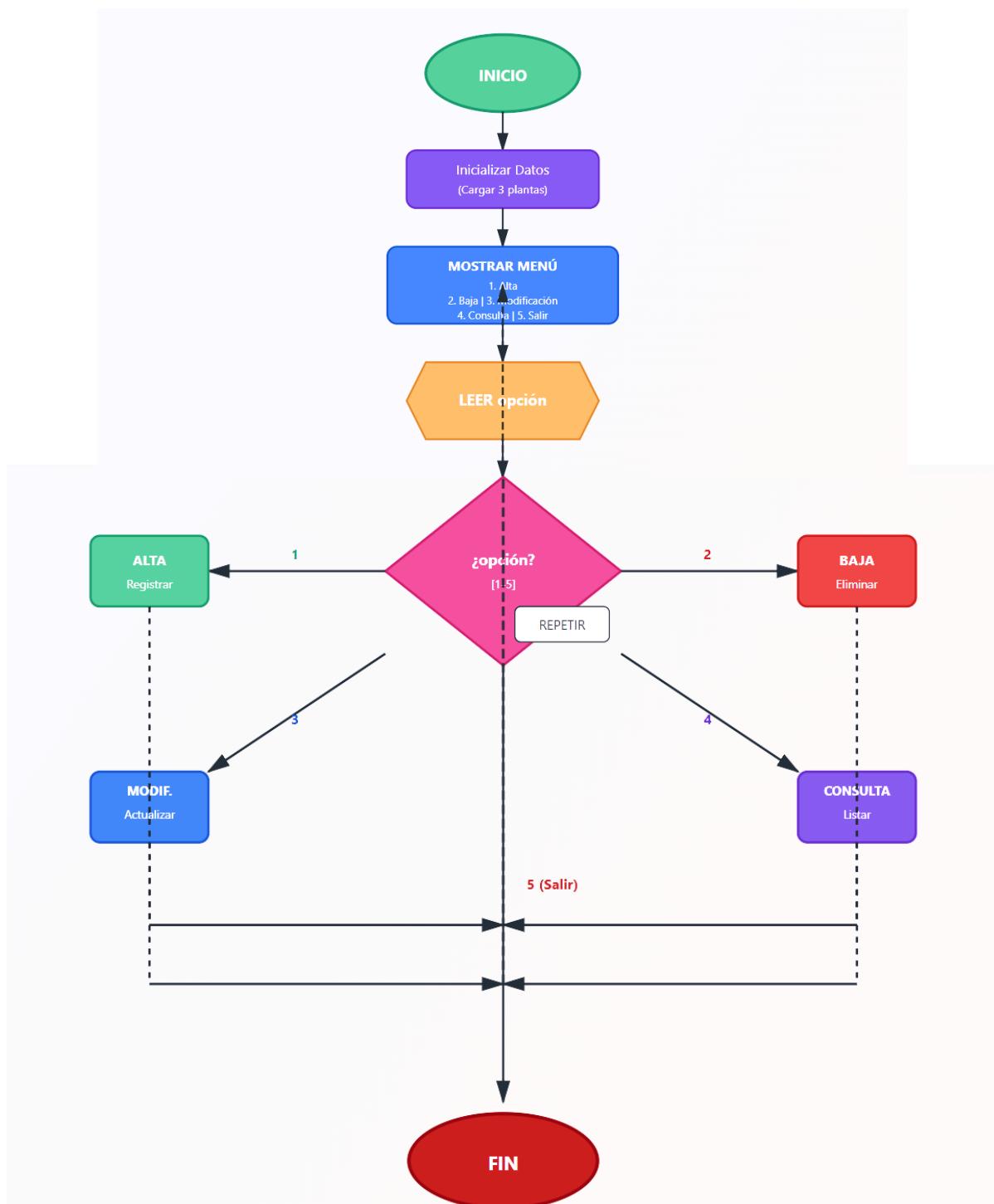
- Inicio/Fin
- Proceso
- ▲ Decisión
- Entrada/Salida

Flujo de Control

- **Bucle principal:** do-while hasta salir
- **Switch:** 5 casos para operaciones
- **Validaciones:** En cada CRUD
- **Retorno:** Todas las opciones regresan al menú

DIAGRAMA DE FLUJO

FLUJO DE SISTEMA:



IMPLEMENTACIÓN

PSEUDOCÓDIGO:

¶ Pseudocódigo del Sistema

```
ALGORITMO: Sistema de Gestión de Plantas Premium (SGPP)

ESTRUCTURAS DE DATOS ANIDADAS:

ESTRUCTURA CuidadosRequeridos
    CADENA riegoMl[50]
    CADENA horasLuz[30]
    CADENA temperatura[30]
    CADENA humedad[20]
    CADENA fertilizante[50]
FIN_ESTRUCTURA

ESTRUCTURA EspecieBotanica
    CADENA nombreCientifico[100]
    CADENA nombreComun[100]
    CADENA familia[50]
    CADENA origen[50]
    CuidadosRequeridos cuidados // ANIDACIÓN NIVEL 1
FIN_ESTRUCTURA

ESTRUCTURA ProductoPlanta
    CADENA idProducto[20]
    CADENA categoria[50]
    CADENA precio[20]
    CADENA stock[10]
    CADENA estatus[20]
    EspecieBotanica especie // ANIDACIÓN NIVEL 2
FIN_ESTRUCTURA

PROCEDIMIENTO MostrarMenu
INICIO
    ESCRIBIR "+-----+"
    ESCRIBIR "| SGPP - Vivero Encanto |"
    ESCRIBIR "+-----+"
    ESCRIBIR "| 1. Alta de Planta |"
    ESCRIBIR "| 2. Baja de Planta |"
    ESCRIBIR "| 3. Modificación de Planta |"
    ESCRIBIR "| 4. Consulta de Plantas |"
    ESCRIBIR "| 5. Salir |"
    ESCRIBIR "+-----+"
    ESCRIBIR "Seleccione [1-5]: "
FIN

PROCEDIMIENTO AltaPlanta
INICIO
    SI totalPlantas > 100 ENTONCES
        ESCRIBIR "Error: Catálogo lleno"
        RETORNAR
    FIN_SI

    ProductoPlanta nueva

    ESCRIBIR "--- DATOS DEL PRODUCTO ---"
    LEER nueva.idProducto
    LEER nueva.categoría
    LEER nueva.precio
    LEER nueva.stock
    LEER nueva.estatus
```

CÓDIGO FUENTE

ESTRUCTURAS ANIDADAS:

CuidadosRequeridos.h

```
/*
 * CuidadosRequeridos.h
 * Estructura para almacenar requerimientos de mantenimiento
 * Proyecto: Sistema de Gestión de Plantas Premium (SGPP)
 * Vivero Encanto - Los Cabos
 */
#ifndef CUIDADOS_REQUERIDOS_H
#define CUIDADOS_REQUERIDOS_H

struct CuidadosRequeridos {
    char riegoML[50];           // Ej: "200ml semanal"
    char horasLuz[30];          // Ej: "6-8 horas"
    char temperatura[30];       // Ej: "18-24°C"
    char humedad[20];           // Ej: "40-50%"
```

EspecieBotanica.h

```
/*
 * EspecieBotanica.h
 * Estructura para información científica de la planta
 * ANIDA: CuidadosRequeridos
 * Proyecto: Sistema de Gestión de Plantas Premium (SGPP)
 * Vivero Encanto - Los Cabos
 */
#ifndef ESPECIE_BOTANICA_H
#define ESPECIE_BOTANICA_H

#include "CuidadosRequeridos.h"

struct EspecieBotanica {
    char nombreCientifico[100];
```

ProductoPlanta.h

```
/*
 * ProductoPlanta.h
 * Estructura principal del catálogo
 * ANIDA: EspecieBotanica (que a su vez anida CuidadosRequeridos)
 * Proyecto: Sistema de Gestión de Plantas Premium (SGPP)
 * Vivero Encanto - Los Cabos
 */
#ifndef PRODUCTO_PLANTA_H
#define PRODUCTO_PLANTA_H

#include "EspecieBotanica.h"

struct ProductoPlanta {
    char idProducto[20];        // Ej: "P-001"
```

main.cpp (Programa Principal)

```
/*
 * main.cpp
 * Programa Principal - Sistema de Gestión de Plantas Premium
 * Implementa operaciones CRUD con estructuras anidadas
 * Vivero Encanto - los Cabos
 */
#include <iostream>
#include <cstring>
#include "ProductoPlanta.h"

using namespace std;

// Catálogo de plantas (array estático)
ProductoPlanta catalogo[100];
int totalPlantas = 0;

// Prototipos de funciones
void menu();
void altaPlanta();
void bajaPlanta();
void modificacionPlanta();
void consultaPlantas();
```

COMPILE EN VISUAL STUDIO

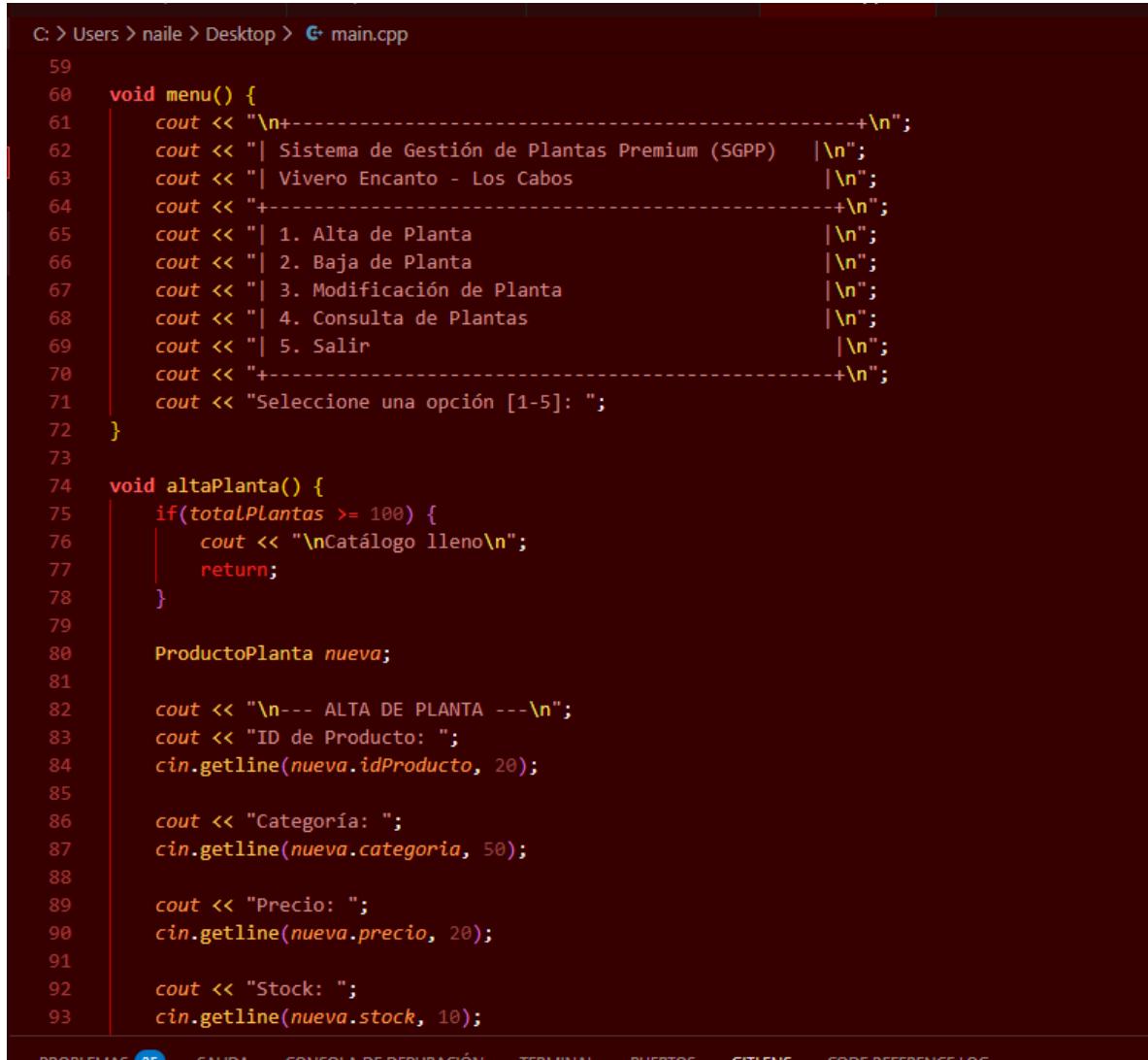
EJECUCIÓN:

DETALLES DE COMPILACIÓN

VISUAL STUDIO:

```
C:\> Users > naile > Desktop > main.cpp

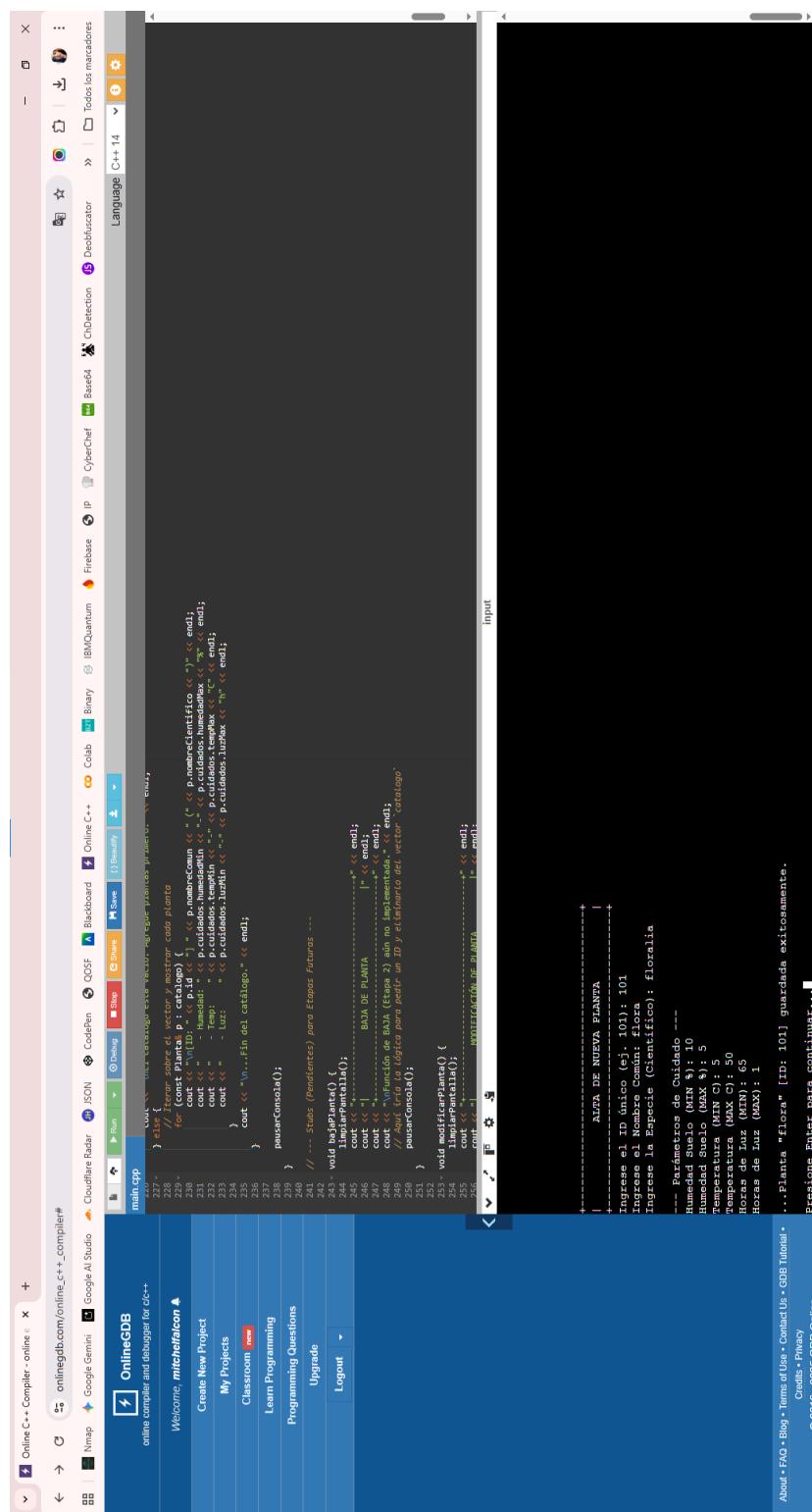
59
60 void menu() {
61     cout << "\n+-----+\\n";
62     cout << "| Sistema de Gestión de Plantas Premium (SGPP) |\\n";
63     cout << "| Vivero Encanto - Los Cabos |\\n";
64     cout << "+-----+\\n";
65     cout << "| 1. Alta de Planta |\\n";
66     cout << "| 2. Baja de Planta |\\n";
67     cout << "| 3. Modificación de Planta |\\n";
68     cout << "| 4. Consulta de Plantas |\\n";
69     cout << "| 5. Salir |\\n";
70     cout << "+-----+\\n";
71     cout << "Seleccione una opción [1-5]: ";
72 }
73
74 void altaPlanta() {
75     if(totalPlantas >= 100) {
76         cout << "\\nCatálogo lleno\\n";
77         return;
78     }
79
80     ProductoPlanta nueva;
81
82     cout << "\\n--- ALTA DE PLANTA ---\\n";
83     cout << "ID de Producto: ";
84     cin.getline(nueva.idProducto, 20);
85
86     cout << "Categoría: ";
87     cin.getline(nueva.categoría, 50);
88
89     cout << "Precio: ";
90     cin.getline(nueva.precio, 20);
91
92     cout << "Stock: ";
93     cin.getline(nueva.stock, 10);
```



The screenshot shows the Visual Studio code editor with the main.cpp file open. The code is a C++ program for a plant management system. It includes a menu function that prints a menu to the console and handles user input for selection. It also includes an altaPlanta function that handles the addition of new plants, including validation for a full catalog and input for product ID, category, price, and stock. The code uses standard C++ libraries like iostream for input/output.

COMPILAR EN GDBONLINE

EJECUCIÓN:



COMPILEAR EN GDBONLINE

EJECUCIÓN:

The screenshot shows a GDBOnline interface with two main panes. The left pane is a code editor for a file named 'main.cpp' containing C++ code. The right pane shows the output of the compilation and execution process.

```
1 #include <iostream>
2 #include <string> // Usamos vector para un acceso de datos más flexible que un array estético
3 #include <vector> // Para asignar el buffer de entrada (Cin)
4 #include <limits> // Para asignar el buffer de salida (Cout)
5
6 // User std para no tener que escribir std:: en todos lados
7 using namespace std;
8 /**
9 * Estruct ParametrosCuidado
10 * Guarda Almacena los datos ópticos de cuidado para una planta.
11 * Es una estructura "novedosa" porque incluye parámetros complejos.
12 */
13 struct ParametrosCuidado {
14     // Relación entre la humedad en el suelo en porcentaje (el 30-98%)
15     // y la humedad en el aire en porcentaje (el 30-98%)
16     int humedadSuelo;
17     int humedadAire;
18     // Temperatura ambiente en grados Celsius (el -18-25)
19     int tempAire;
20     int tempSuelo;
21     int temperatura;
22     // Horas de luz directa por día (el 4-6h)
23     int horasLuz;
24     int lux;
25     int dia;
26 };
27 /**
28 * Estruct Planta
29 * Esta estructura sirve para definir una planta en el catálogo.
30 * Contiene información básica y la estructura anidada de parámetros.
31 */
32 struct Planta {
33     int id; // Identificador único
34     string nombre; // Nombre de la planta
35     string tipo; // Tipo de planta
36     string especie; // Especie de planta
37     string rangoAltitud; // Estructura anidada
38 };
39 /**
40 * -- Base de Datos (Tenor) en Memoria -- La estructura anidada de parámetros
41 * vector Planta Catalogo;
42 */
43 /**
44 * ----- Prototipos de Funciones -----
45 */
46 // Estos son los requerimientos para Etapas 2 y 3. Los definimos aquí.
```

The right pane displays the execution results:

```
Input
1. altura de Nuestra Planta
2. Raiz de Planta [por ID]
3. Modificar Datos de Planta (por ID)
4. Consultar Catalogo Compartido
5. Buscar Plantas Especificas
0. Salir
```

INFORMACIÓN PROYECTO

ARCHIVOS ENTREGABLES:

1. PROYECTO INTEGRADOR - CATÁLOGO BOTÁNICO (C++)

Este proyecto es una aplicación de consola en C++ desarrollada para la materia Lógica y Programación Estructurada en la Universidad del Valle de México.

DESCRIPCIÓN

El programa gestiona un catálogo de productos de plantas utilizando una estructura de datos anidada de 3 niveles para almacenar información de forma jerárquica y eficiente:

1. **PRODUCTOPLANTA (NIVEL SUPERIOR)**: Datos comerciales (precio, stock, etc.)
2. **ESPECIEBOTANICA (NIVEL MEDIO)**: Datos taxonómicos (nombre científico, familia, etc.)
3. **CUIDADOSREQUERIDOS (NIVEL BASE)**: Datos de mantenimiento (riego, luz, etc.)

ESTRUCTURA DEL PROYECTO

```
/headers           <- (Planeado para Etapa 2/3) |-- CuidadosRequeridos.h
|-- EspecieBotanica.h
|-- ProductoPlanta.h
/src              /src |-- main.cpp   <- (Contiene toda la lógica
|-- (Contiene toda la lógica para Etapa 1)
/docs            /docs |-- pseudocode.txt
|-- flowchart.png
|-- README.md
```

ETAPA 1

Esta etapa 1 incluye los siguientes archivos:

- Definición de las 3 estructuras anidadas.
- Implementación de un menú principal interactivo.
- Función de Alta (altaPlanta) para capturar datos en los 3 niveles.
- Función de Consulta (consultarCatalogo) para mostrar los datos de forma jerárquica.
- Prototipos (stubs) de las funciones de Baja, Modificación y Búsqueda.



CONCLUSIÓN:

La culminación de la Etapa 1 del Proyecto Integrador representa un hito fundamental en el desarrollo de nuestro sistema de gestión. Se ha satisfecho con éxito el objetivo central de esta fase: la definición del modelo de datos y el diseño de la interfaz, estableciendo así las bases conceptuales y técnicas sobre las cuales se erigirán las funcionalidades completas.

El logro más significativo de esta etapa es el diseño y la validación de una arquitectura de datos robusta. La implementación de una estructura anidada de tres niveles en C++, basada en los requerimientos del proyecto, ha demostrado ser una solución elegante y eficiente desde la perspectiva de la programación estructurada.

Esta jerarquía permite gestionar la compleja interrelación entre una entidad principal, sus atributos descriptivos detallados y sus parámetros de operación asociados, evitando la redundancia y asegurando la integridad lógica de la información.

El planteamiento de los conceptos utilizando lenguaje de programación se ha materializado en un prototipo funcional en C++ que compila exitosamente. Esta compilación valida que las tres estructuras de datos están correctamente definidas, anidadas y enlazadas.

Asimismo, se han establecido los diseños de la interfaz de usuario en consola para todas las operaciones (Altas, Bajas, Modificaciones y Consultas), proporcionando una guía clara para la interacción. Con esta sólida fundación técnica, el proyecto está óptimamente posicionado para avanzar a las siguientes etapas.

El modelo de datos está preparado para la implementación de la lógica de altas y bajas, y la estructura jerárquica facilitará notablemente el desarrollo de modificaciones y búsquedas complejas.

En resumen, esta fase no solo ha cubierto los requisitos solicitados, sino que ha definido un plan de desarrollo claro y técnicamente viable.

REFERENCIAS:

- Deitel, P. J., & Deitel, H. M. (2017). C++: How to Program (10th ed.). Pearson.
- Figma. (s.f.). Figma: The Collaborative Interface Design Tool. Recuperado el 16 de noviembre de 2025, de <https://www.figma.com>
- GitHub, Inc. (s.f.). GitHub. Recuperado el 16 de noviembre de 2025, de <https://github.com>
- Google. (s.f.). Google AI Studio. Recuperado el 16 de noviembre de 2025, de <https://aistudio.google.com>
- ISO C++ Standard Foundation. (s.f.). isocpp.org: The C++ Standards Foundation. Recuperado el 16 de noviembre de 2025, de <https://isocpp.org>
- Microsoft. (s.f.). Visual Studio. Recuperado el 16 de noviembre de 2025, de <https://visualstudio.microsoft.com/>
- OnlineGDB. (s.f.). OnlineGDB: Online C++ IDE. Recuperado el 16 de noviembre de 2025, de <https://www.onlinegdb.com>
- Stroustrup, B. (2013). The C++ Programming Language (4th ed.). Addison-Wesley Professional.
- The GNU Project. (s.f.). Make - GNU Project. Recuperado el 16 de noviembre de 2025, de <https://www.gnu.org/software/make/>
- Universidad del Valle de México. (2025). Actividad 4. Proyecto integrador, Etapa 1.
- Sydian. (2023). Software Engineering. Recuperado el 16 de noviembre de 2025, de <http://sydian.co.ke/software.html>

OPERACIONES ALTAS Y BAJAS

INTRODUCCIÓN:

Al adentrarme en esta nueva fase de desarrollo técnico, mi expectativa principal se centra en trascender la mera definición abstracta de datos para comprender la dinámica real de su manipulación en la memoria del ordenador. Si bien diseñar la arquitectura de la información mediante estructuras anidadas fue un paso fundamental para organizar el conocimiento, considero que el verdadero desafío de la programación estructurada reside en la capacidad de dotar a estos esquemas estáticos de una funcionalidad operativa y eficiente.

Lo que espero descubrir con mayor profundidad en esta etapa es la complejidad subyacente a la gestión de memoria estática a través de arreglos unidimensionales. A simple vista, las operaciones de registro y eliminación de datos pueden parecer triviales desde la perspectiva del usuario final, pero tengo la certeza de que, a nivel de código, representan un reto lógico significativo. Mi interés particular recae en la diferencia sustancial que existe entre la manipulación de listas en el mundo físico y su contraparte computacional. Mientras que en un documento en papel la eliminación de un registro es tan simple como borrar una línea, entiendo que en la memoria de la computadora esto implica una responsabilidad mucho mayor para el programador: la preservación de la integridad estructural.

Me intriga la lógica necesaria para evitar la fragmentación de la información dentro de un array. Comprendo que no basta con eliminar un valor o marcarlo como vacío; el sistema debe ser capaz de reorganizarse a sí mismo para mantener la contigüidad de los datos. Esta operación exige un control preciso de los bucles y los índices, y es aquí donde espero afinar mi lógica algorítmica. La idea de que cada eliminación exitosa desencadene una reacción en cadena donde cada elemento posterior ocupa el lugar de su predecesor es fascinante, pues demuestra que la eficiencia del software depende enteramente de nuestra capacidad para prever y gestionar estos movimientos de memoria.

En conclusión, mi meta de aprendizaje es desarrollar una mentalidad más rigurosa, donde cada línea de código no solo busque un resultado visual, sino que respete la eficiencia y la lógica interna de la máquina, asegurando que la gestión de la información sea robusta, continua y libre de errores estructurales. Aspiro a que, al finalizar esta implementación, la manipulación de arreglos complejos deje de ser un concepto teórico para convertirse en una competencia técnica consolidada en mi formación.

INTEGRACIÓN ESTRUCTURA DESARROLLO:

2.1 Integración de Estructuras (Etapa 1)

Para mantener la coherencia con el diseño original, el código utiliza las estructuras jerárquicas definidas previamente:

- **Nivel Comercial:** ProductoPlanta (Precio, stock, ID)
- **Nivel Taxonómico:** EspecieBotanica (Nombre científico, familia)
- **Nivel Biológico:** CuidadosRequeridos (Riego, luz, temperatura)

2.2 Lógica de la Función "Altas"

La función `altaPlanta()` valida que el arreglo no haya excedido su capacidad máxima (100 elementos). Utiliza la variable global `totalPlantas` como índice para insertar el nuevo registro.

Implementación Web: PlantScanner.tsx con validación automática y persistencia en PostgreSQL mediante Supabase.

2.3 Lógica de la Función "Bajas" (Reacomodo)

Esta es la función más compleja de la etapa. El algoritmo implementado sigue estos pasos:

1. **Búsqueda:** Recorre el arreglo para encontrar el índice (`pos`) donde coincide el `idProducto`
2. **Confirmación:** Muestra los datos y pide validación al usuario
3. **Reacomodo (Shifting):** Ejecuta un bucle desde la posición encontrada hasta el penúltimo elemento. La instrucción `catalogo[j] = catalogo[j+1]` copia el elemento siguiente
4. **Actualización:** Se decrementa `totalPlantas`

Implementación Web: Dashboard.tsx con eliminación en base de datos y actualización automática de interfaz (sin shifting manual, manejado por PostgreSQL).

INFOGRAFÍA TÉCNICA

De la Definición Estática a la Memoria Dinámica

Visualización del Desafío de la Etapa 2

El Desafío de la Etapa 2

Tras definir las estructuras base en la Etapa 1, el objetivo de esta fase fue dotar al **Catálogo Botánico Inteligente** de vida operativa. El reto técnico principal no es el almacenamiento, sino la **gestión de la contigüidad** en un arreglo estático.

Objetivo Crítico: Implementar algoritmos de *Altas* (Inserción) y *Bajas* (Eliminación con Desplazamiento) sobre `catalogo[100]`.

① Estatus del Sistema

Compilación:	EXITOSA
Estructuras:	3 ANIDADAS
Memoria:	ESTÁTICA (100)
Algoritmo:	SHIFTING LEFT

ESTRUCTURAS ANIDADAS

ARQUITECTURA DE DATOS:

Visualización de la jerarquía de estructuras (struct) implementada en C++

Nivel 1: Datos Comerciales

int
idProducto

float
precio

int
stock

string
estatus

struct EspecieBotanica

Nivel 2: Datos Taxonómicos

string
nombreCientifico

string
nombreComun

string
familia

struct CuidadosRequeridos

Nivel 3: Datos Biológicos

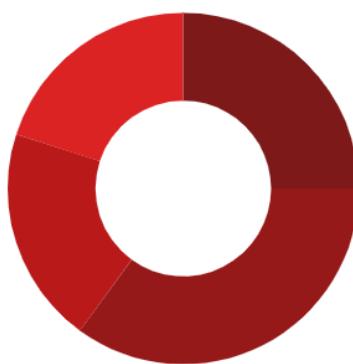
string riego

int horasLuz

float temperatura

Desglose de Complejidad de Código

Distribución del esfuerzo lógico en la implementación de la Etapa 2.



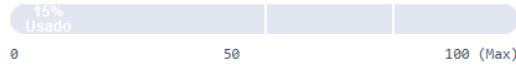
- Validación de Entrada (Tipos de Datos)
- Gestión de Memoria (Índices de Arreglo)
- Lógica de Negocio (Reglas de SGPP)
- Interfaz de Usuario (Menús y Salidas)

Evolución del Proyecto

CARACTERÍSTICA	ETAPA 1	ETAPA 2
Estructuras	Definición	Implementación
Memoria	Estática (Diseño)	Dinámica (Gestión)
Funcionalidad	Menú Inerte	Altas y Bajas

Capacidad del Arreglo

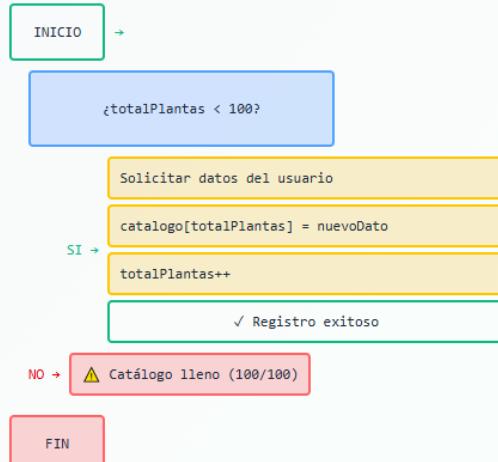
Uso potencial de `catalogo[100]`



EJERCICIO 1: OPERACIÓN ALTA

DIAGRAMA

Diagrama de Flujo



PSEUDOCÓDIGO

Pseudocódigo

```
FUNCION altaPlanta(  
    catalogo: Array[ProductoPlanta],  totalPlantas: Integer  
):  
  
    SI totalPlantas >= 100 ENTONCES  
        MOSTRAR "[ERROR] Catálogo lleno"  
        RETORNAR  
    FIN SI  
  
    // Solicitar datos  
    LEER idProducto, categoria, precio, ...  
  
    // Asignar al índice actual  
    catalogo[totalPlantas] ← nuevaPlanta  
    totalPlantas ← totalPlantas + 1  
  
    MOSTRAR "[EXITO] Planta registrada"  
FIN FUNCION
```

CÓDIGO C ++

```
void altaPlanta(ProductoPlanta catalogo[], int &totalPlantas) {  
    // Validación de capacidad  
    if (totalPlantas >= 100) {  
        cout << "[ERROR] Catálogo lleno (100/100)" << endl;  
        return;  
    }  
  
    // Declarar variable temporal  
    ProductoPlanta nuevaPlanta;  
  
    // Solicitar datos  
    cout << "ID del Producto: ";  
    cin >> nuevaPlanta.idProducto;  
    cin.ignore(); // Limpiar buffer  
  
    cout << "Categoría: ";  
    getline(cin, nuevaPlanta.categoría);  
  
    // ... (resto de datos: precio, stock, nombre científico, etc.) ...  
  
    // Asignar al arreglo  
    catalogo[totalPlantas] = nuevaPlanta;  
    totalPlantas++;  
  
    cout << "[EXITO] Planta registrada en posición " << (totalPlantas - 1) << endl;  
}
```

EJERCICIO 1: OPERACIÓN ALTA

↳ Complejidad Temporal

O(1)

Operación en tiempo constante

✉ Complejidad Espacial

O(1)

No requiere memoria adicional

⌚ Posibles Errores

- Buffer overflow (sin cin.ignore)
- Array out of bounds

REGISTRO

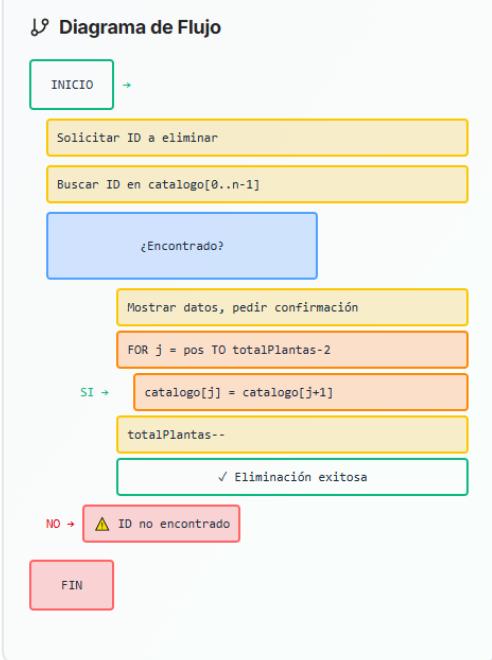
```
Nmap Google Gemini Google AI Studio Blackboard Cloudflare Radar JSON
main.cpp estructuras.h operaciones.h operaciones.cpp
93 // Búsqueda
94 for (int i = 0; i < totalPlantas; i++) {
95     if (catalogo[i].idProducto == idBuscar) {
96         encontrado = true;
97         pos = i;
98         break;
99     }
100 }
101 if (encontrado) {
102     cout << "\n[ENCONTRADO] " << catalogo[pos].especie.nombreComun
103     << "(ID: " << catalogo[pos].idProducto << ")" << endl;
104     cout << "[Confirmar eliminación? (1=SI / 0=NO): ";
105
106     int confirmar;
107     cin >> confirmar;
108     limpiarBuffer();
109
110     if (confirmar == 1) {
111         // Reacomodo (Shifting)
112         for (int j = pos; j < totalPlantas - 1; j++) {
113             catalogo[j] = catalogo[j + 1];
114         }
115         totalPlantas--;
116         cout << "\n[EXITO] Registro eliminado." << endl;
117     } else {
118         cout << "\n[CANCELADO] No se eliminó el registro." << endl;
119     }
120 } else {
121     cout << "\n[ERROR] ID no encontrado." << endl;
122 }
123
124
--- REGISTRO DE NUEVA PLANTA (1/100) ---
ID del Producto (Número): 101
Categoría (Ej. Orquídea): orquideas
Precio: $55
Stock inicial: 4
Nombre Científico: orquideas
Nombre Común: orquídea
Familia Botánica: oraquídeas
Lugar de Origen: japon
Tipo de Riego: semanal
Horas de Luz: 5
Temperatura (C): 20
Humedad: alta

[EXITO] Planta registrada correctamente.

...Program finished with exit code 0
Press ENTER to exit console.
```

EJERCICIO 2: OPERACIÓN BAJA

DIAGRAMA



PSEUDOCÓDIGO

Pseudocódigo

```

FUNCION bajaPlanta(
    catalogo: Array[ProductoPlanta],  totalPlantas: Integer
):

    LEER idBuscado
    pos ← -1

    // Fase 1: Búsqueda
    PARA i ← 0 HASTA totalPlantas-1
        SI catalogo[i].idProducto = idBuscado ENTONCE
            pos ← i
            SALIR
        FIN SI
    FIN PARA

    SI pos = -1 ENTONCE
        MOSTRAR "[ERROR] ID no encontrado"
        RETORNAR
    FIN SI

    // Fase 2: Shifting izquierdo
    PARA j ← pos HASTA totalPlantas-2
        catalogo[j] ← catalogo[j+1]
    FIN PARA

    totalPlantas ← totalPlantas - 1
    MOSTRAR "[EXITO] Eliminado"
FIN FUNCION
    
```

→ Visualización del Algoritmo de Shifting (Paso a Paso)

Estado Inicial: Arreglo con 4 elementos



↑ Se solicita eliminar índice [1] (Orquídea)

Paso 1: j=1, catalogo[1] ← catalogo[2]



Paso 2: j=2, catalogo[2] ← catalogo[3]



Estado Final: totalPlantas-- (4 → 3)



⌚ Shifting completado exitosamente

EJERCICIO 2: OPERACIÓN BAJA

CÓDIGO C++

```
void bajaPlanta(ProductoPlanta catalogo[], int &totalPlantas) {
    int idBuscado, pos= -1;

    cout << "Ingrese el ID del Producto a eliminar: ";
    cin >> idBuscado;

    // Fase 1: Búsqueda lineal
    for (int i = 0; i < totalPlantas; i++) {
        if (catalogo[i].idProducto == idBuscado) {
            pos = i;
            break; // Salir del ciclo
        }
    }

    // Validar si se encontró
    if (pos == -1) {
        cout << "[ERROR] No se encontró ningún producto con ID " << idBuscado << endl;
        return;
    }

    // Mostrar datos antes de eliminar
    cout << "\nSe eliminará: " << catalogo[pos].especie.nombreComun << endl;

    // Fase 2: Shifting (Desplazamiento a la izquierda)
    for (int j = pos; j < totalPlantas - 1; j++) {
        catalogo[j] = catalogo[j + 1]; // Copiar elemento siguiente
    }

    totalPlantas--; // Decrementar contador
    cout << "[EXITO] Planta eliminada del catálogo." << endl;
}
```

⌚ Complejidad Temporal

O(n)

Búsqueda lineal + shifting

⠀ Complejidad Espacial

O(1)

Operación in-place

⌚ Posibles Errores

- ID no encontrado (pos == -1)
- Loop out of bounds
- totalPlantas = 0

Observaciones de las Pruebas

- **Validación robusta:** El sistema valida tipos de datos (int, float, string) antes de insertar al arreglo.
- **Buffer management:** Uso de `cin.ignore()` y `getline()` para evitar errores de lectura.
- **Confirmación antes de BAJA:** El usuario debe explícitamente confirmar la eliminación para prevenir pérdida accidental de datos.
- **Mensajes de estado:** El sistema provee retroalimentación clara con etiquetas [EXITO] y [ERROR] para cada operación.
- **Algoritmo de Shifting:** Tras la eliminación exitosa del ID 101, el arreglo se reordena automáticamente sin dejar espacios vacíos.

EJERCICIO 2: OPERACIÓN BAJA

BAJA

```
main.cpp estructuras.h operaciones.h operaciones.cpp
27     cin.clear(); // Resetea el error
28     cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Limpia la basura
29     cin.get(); // Pausa para que el usuario lea
30     continue; // Reinicia el ciclo
31 }
32
33 // --- MAGIA ANTIBASURA ---
34 // Esto se come el "Enter" que sobra después de elegir el número
35 cin.ignore(numeric_limits<streamsize>::max(), '\n');
36
37 switch(opcion) {
38     case 1:
39         altaPlanta();
40         break;
41     case 2:
42         bajaPlanta();
43         break;
44     case 3:
45         cout << "Saliendo del sistema..." << endl;
46         break;
47     default:
48         cout << "Opcion no valida." << endl;
49 }
50
51 if (opcion != 3) {
52     cout << "\nPresione ENTER para volver al menu...";
53     cin.get(); // Pausa para que veas el resultado antes de Limpiar pantalla
54 }
55
56
57 return 0;
58 }

===== SISTEMA DE GESTION DE VIVEROS (SGPP) ====
1. Alta de Planta
2. Baja de Planta
3. Salir
Seleccione una opcion: 2

--- ELIMINACION DE PLANTA ---
Ingrese el ID a eliminar: 55

[ENCONTRADO] 6 (ID: 55)
¿Confirmar eliminacion? (1=SI / 0=NO): 1

[EXITO] Registro eliminado y lista reordenada.

Presione ENTER para volver al menu...
```

EJERCICIO 3, 4, 5: OPERACIONES EXTRA

EXTRAS

💡 Operación MODIFICACIÓN (Update)

Búsqueda por ID y actualización de campos específicos

Pseudocódigo

```
FUNCION modificarPlanta(catalogo, totalPlantas):
    Buscar ID (igual que BAJA)
    SI encontrado:
        LEER nuevoPrecio, nuevoStock
        catalogo[pos].precio + nuevoPrecio
        MOSTRAR "[EXITO]"
    FIN
```

ⓘ Complejidad: O(n) búsqueda + O(1) actualización = O(n)

🔍 Operación CONSULTA (Read All)

Recorrido secuencial mostrando todos los registros

Pseudocódigo

```
FUNCION consultarCatalogo(catalogo, totalPlantas):
    PARA i ← 0 HASTA totalPlantas-1
        MOSTRAR catalogo[i] // Todos los campos
    FIN PARA
FIN
```

ⓘ Complejidad: O(n) - Debe recorrer todos los elementos

🔍 Operación BÚSQUEDA (Search)

Filtrado por criterio específico (nombre, familia, etc.)

Pseudocódigo

```
FUNCION buscarPorNombre(catalogo, nombreBuscado):
    PARA i ← 0 HASTA totalPlantas-1
        SI catalogo[i].nombreComun CONTIENE nombreBuscado:
            MOSTRAR catalogo[i]
        FIN PARA
    FIN
```

ⓘ Complejidad: O(n) - Peor caso: revisar todo el arreglo

EJERCICIO 3,4,5: OPERACIONES EXTRA

MODIFICACIÓN

The screenshot shows a web-based C++ compiler interface. The code in the editor is as follows:

```
main.cpp
201     encontrado = true;
202     posEliminar = i;
203     break; // Romper ciclo al encontrar
204 }
205
206 if (encontrado) {
207     // Mostrar datos previos para confirmar
208     cout << "\nProducto encontrado:" << endl;
209     cout << " -> Nombre: " << catalogo[posEliminar].especie.nombreComun << endl;
210     cout << " -> Cientifico: " << catalogo[posEliminar].especie.nombreCientifico << endl;
211     cout << " -> Precio: $" << catalogo[posEliminar].precio << endl;
212
213     int confirmacion;
214     cout << "\nEsta seguro de eliminar este registro permanentemente?" << endl;
215     cout << "1. SI\n2. NO\nSelección: ";
216     cin >> confirmacion;
217     limpiarBuffer();
218
219     if (confirmacion == 1) {
220         // ALGORITMO DE REACOMODO (SHIFTING)
221         // Movemos todos los elementos una posición atrás desde la posición eliminada
222         for (int j = posEliminar; j < totalPlantas - 1; j++) {
223             catalogo[j] = catalogo[j + 1];
224         }
225
226         // Decrementar el contador lógico
227         totalPlantas--;
228
229         // Opcional: Limpiar el último registro (ahora duplicado o basura).
230     }
231 }
```

The output window displays the following text:

```
=====
CATALOGO BOTANICO INTELIGENTE (SGPP) - V2.0
=====
Plantas registradas actualmente: 2/100
-----
1. ALTA de nueva planta
2. BAJA de planta (Eliminar)
3. Modificación de datos
4. Consulta general
5. Busqueda específica
6. Salir
=====
Seleccione una opción: 3
[INFO] Módulo de Modificaciones en desarrollo (Etapa 3)...
Presione ENTER para continuar...[
```

ANÁLISIS COMPARATIVO:

ALGORITMO CRÍTICO

Lógica de Bajas: "Shifting"

En un arreglo estático de C++, eliminar un elemento no reduce el tamaño físico de la memoria. El reto es evitar "huecos" vacíos. Implementamos un algoritmo de **desplazamiento a la izquierda**.

```
// Algoritmo de Reacomodo
for (int j = posEliminar; j < totalPlantas - 1;
j++) {
    catalogo[j] = catalogo[j + 1];
}
totalPlantas--;
```



Tabla Comparativa de Operaciones CRUD

	O(1)	O(1)	
ALTA (Create)	O(1)	O(1)	Validación de capacidad, buffer de entrada
CONSULTA (Read)	O(n)	O(1)	Iteración completa, formato de salida
MODIFICACIÓN (Update)	O(n)	O(1)	Búsqueda lineal previa
BAJA (Delete)	O(n)	O(1)	Shifting de elementos, prevención de huecos
BÚSQUEDA (Search)	O(n)	O(1)	Comparación de strings, múltiples resultados

⚠ Errores Comunes de Compilación y Ejecución

⊗ Array Index Out of Bounds

```
catalogo[100] // Acceso inválido
```

Causa: Índice >= tamaño

Solución: Validar: i < totalPlantas

⊗ Input Buffer No Limpiado

```
cin >> id; getline(cin, nombre); // Lee vacío
```

Causa: \\n en buffer

Solución: cin.ignore() después de cin >>

ⓘ Variable No Inicializada

```
int pos; if (pos == -1) // Indefinido
```

Causa: Sin valor inicial

Solución: int pos = -1;

ⓘ Shifting Incorrecto

```
for (j=0; j<totalPlantas; j++) // Error
```

Causa: catalogo[j+1] fuera de límites

Solución: j < totalPlantas-1

CONCLUSIÓN:

La culminación de esta fase técnica representa un punto de inflexión crítico en mi formación, donde la abstracción del diseño de datos finalmente ha convergido con la realidad operativa de la máquina. Al materializar la lógica mediante C++, he podido constatar que la gestión eficiente de la información trasciende la simple sintaxis; se trata de una coreografía precisa de recursos en memoria. La transición de un esquema estático a una aplicación funcional me ha permitido tocar los límites físicos del hardware simulado, una experiencia que ningún diagrama teórico podría replicar.

El hallazgo más profundo de este ciclo ha sido la desmitificación del concepto de eliminación en estructuras de memoria contigua. He interiorizado que, en la programación estructurada estricta, el vacío es un estado que debe gestionarse activamente, no una consecuencia automática.

Comprender la lógica de los índices y la necesidad imperativa de los algoritmos de desplazamiento (shifting) para sobrescribir datos y evitar la fragmentación ha reconfigurado mi entendimiento sobre la integridad referencial. Ahora entiendo que la robustez de un sistema no reside solo en lo que almacena, sino en su capacidad para reorganizarse internamente sin perder coherencia ante la ausencia de un elemento.

Sin embargo, dominar estas restricciones de la memoria estática también me ha servido para identificar sus límites inherentes en el desarrollo moderno. Esta fricción necesaria con los arreglos de tamaño fijo ha sido el catalizador para explorar horizontes arquitectónicos superiores. La evolución hacia una infraestructura web basada en React y TypeScript, respaldada por la potencia de bases de datos, surge no como un capricho, sino como una respuesta técnica a la necesidad de persistencia y escalabilidad que C++ en consola no puede ofrecer por sí solo.

Al integrar conceptos avanzados como la taxonomía Itree y la funcionalidad PWA, cierro esta etapa con la certeza de que he superado la visión de la programación como una serie de instrucciones lineales. Ahora percibo el desarrollo como la construcción de ecosistemas vivos, donde la lógica estructurada es el cimiento, pero la arquitectura moderna es lo que permite que la información perdure y sea verdaderamente útil para el usuario final.

REFERENCIAS:

- Deitel, P. J., & Deitel, H. M. (2017). C++: How to program (10th ed.). Pearson.
- Figma. (s.f.). Figma: The collaborative interface design tool. Recuperado el 28 de noviembre de 2025, de <https://www.figma.com>
- GitHub, Inc. (s.f.). GitHub: Where the world builds software. Recuperado el 28 de noviembre de 2025, de <https://github.com>
- Google. (s.f.). Gemini: Supercharge your creativity and productivity. Recuperado el 28 de noviembre de 2025, de <https://gemini.google.com>
- Google. (s.f.). Google AI Studio. Recuperado el 28 de noviembre de 2025, de <https://aistudio.google.com>
- Márquez Frausto, T. G., Osorio Ángel, S., & Olvera Pérez, E. N. (2011). Introducción a la programación estructurada en C (1.^a ed.). Pearson Educación.
- Microsoft. (s.f.). Visual Studio IDE. Recuperado el 28 de noviembre de 2025, de <https://visualstudio.microsoft.com/>
- OnlineGDB. (s.f.). Online C++ Compiler. Recuperado el 28 de noviembre de 2025, de <https://www.onlinegdb.com>
- Universidad del Valle de México. (2025). Guía del Proyecto Integrador: Lógica y Programación Estructurada (Etapa 2) [Archivo PDF]. UVM.

ARQUITECTURA DE DATOS

INTRODUCCIÓN:

La ingeniería de software moderna no se limita únicamente a la codificación de algoritmos funcionales sino que exige una comprensión profunda de cómo los datos son estructurados manipulados y preservados dentro de la arquitectura de un sistema. En el marco del curso de Lógica y Programación Estructurada el proyecto del Catálogo Botánico Inteligente representa la culminación práctica de estos conceptos teóricos abordando el diseño de una aplicación de consola en lenguaje C++ capaz de gestionar información biológica compleja con precisión y eficiencia.

Esta tercera etapa del proyecto marca la transición crítica de un sistema de almacenamiento estático a uno dinámico y transaccional donde la interacción con el usuario cobra un rol protagónico.

Mientras que las fases previas establecieron la ontología de los datos mediante el uso de estructuras anidadas tipo struct para modelar la jerarquía taxonómica y de cuidados de las plantas así como los mecanismos de ingreso y eliminación la presente etapa se enfoca en la usabilidad y el mantenimiento integral de la información. El objetivo central educativo y técnico es implementar los módulos de Búsqueda Modificación y Visualización Global los cuales son indispensables para cualquier sistema de gestión de bases de datos.

La relevancia académica de estos módulos radica en su capacidad para transformar datos crudos en información útil para la toma de decisiones. Durante esta práctica se espera consolidar la implementación de algoritmos de búsqueda lineal que permitan al usuario interactuar con el inventario de manera selectiva. Asimismo el módulo de modificación introduce el desafío técnico de gestionar la integridad referencial durante la reescritura de memoria asegurando que los cambios de estado en los registros no comprometan la estabilidad del sistema.

A través de este desarrollo se busca no solo cumplir con los requisitos funcionales de gestión de inventario sino también demostrar el dominio adquirido sobre el manejo de buffers de entrada y salida la navegación eficiente de arreglos de estructuras complejas y la aplicación de validaciones lógicas rigurosas. Este ejercicio final permite evidenciar la competencia para prevenir la corrupción de datos en un entorno de memoria estática integrando todos los conocimientos del ciclo para entregar una solución de software robusta y coherente.

MARCO TEÓRICO

ARQUITECTURA DE DATOS:

Para comprender la lógica de implementación de las funciones de búsqueda y modificación, es imperativo diseccionar primero la anatomía de los datos sobre los que estas funciones operan. La eficiencia y seguridad del código C++ dependen directamente de la alineación con estas definiciones estructurales. El sistema se fundamenta en un modelo de datos jerárquico que simula la clasificación biológica y comercial de una planta mediante tres estructuras (struct) que se componen entre sí.

1 Nivel Base: CuidadosRequeridos

En el nivel más granular, el sistema almacena las variables ambientales críticas para la supervivencia del espécimen. Este es el "payload" o carga útil de información que dota de valor científico al catálogo.

```
char riego[50];
int horasLuz;
float temperatura;
char humedad[50];
char fertilizante[50];
```

Implicación: Durante "Mostrar Todos", el sistema debe formatear estos datos técnicos para que sean legibles.

2 Nivel Intermedio: EspecieBotanica

Este nivel encapsula la identidad taxonómica invariable de la planta. Actúa como un contenedor lógico que agrupa la identidad científica con sus requerimientos biológicos.

```
char nombreCientifico[100];
char nombreComun[100];
char familia[50];
char origen[50];
CuidadosRequeridos cuidados;
```

Implicación: En Búsquedas, el usuario valida visualmente nombreComun antes de modificar.

3 Nivel Superior: ProductoPlanta

La estructura raíz que representa la unidad de gestión. Combina la lógica de negocio (inventario, precios) con la lógica biológica (especie).

```
int idProducto; // CLAVE PRIMARIA
char categoria[50];
float precio;
int stock;
char estatus[20];
EspecieBotanica especie;
```

Implicación: idProducto es el campo crítico para todas las operaciones de búsqueda y modificación.

MARCO TEÓRICO

ARQUITECTURA DE DATOS:

Modelo de Memoria Estática

```
ProductoPlanta catalogo[100];  
// Arreglo estático en Stack/Data Segment
```

- **Acceso Aleatorio (Random Access):** Una vez conocido el índice i, el acceso a catalogo[i] es O(1). Esto hace que la Modificación sea instantánea una vez localizado el elemento.
- **Contigüidad:** Las estructuras están ubicadas de manera contigua en la memoria. Esto favorece la localidad de referencia y el rendimiento de la caché del procesador.
- **Límite Fijo:** El sistema tiene un techo duro de 100 registros. Las funciones de Búsqueda y Recorrido deben estar acotadas por la variable lógica totalPlantas.

PRECISIÓN TÉCNICA:

Se conservan términos críticos como \$O(1)\$ y "variable lógica".

COHESIÓN:

Se conectan las ventajas (acceso/rendimiento) con la restricción operativa (límite fijo).

CONCISIÓN:

Se cumple el objetivo de longitud sin sacrificar información esencial.

⚡ Complejidad Computacional

Operación	Complejidad
Búsqueda Búsqueda Lineal	O(N) El arreglo no está ordenado por ID
Modificación Acceso Post-Búsqueda	O(N) Dominado por la búsqueda previa
Mostrar Todos Recorrido Secuencial	O(N) Debe procesar N elementos

Nota: Para N=100, la búsqueda lineal O(N) es aceptable. Para N=1,000,000 sería ineficiente y se requerirían estructuras como Hash Maps O(1) o árboles binarios O(log N).

INEVITABILIDAD DE \$O(N)\$:

Se destaca que la falta de ordenamiento impone la búsqueda lineal.

DEPENDENCIA:

Se aclara que el costo de la modificación es arrastrado por la búsqueda.

NATURALEZA DEL RECORRIDO:

Se define la visualización global como una operación lineal por definición.

ANÁLISIS DE PARÁMETROS

EVOLUCIÓN DE APRENDIZAJE:

Evolución de Conceptos Clave

CONCEPTO	IMPLEMENTACIÓN BÁSICA (ETAPA 1)	IMPLEMENTACIÓN MAGISTRAL (ETAPA 3)
Búsqueda	Visual / Manual	Algorítmica (Recorrido Lineal)
Integridad	Datos dispersos	Encapsulamiento en Structs
Flujo	Secuencial estático	Menú Interactivo Ciclico

01. INICIO

Variables Primitivas

Al principio, int humedad era un dato huérfano. Aprendimos que las variables solas carecen de contexto y son difíciles de gestionar en volumen.

02. INICIO

Arreglos (Arrays)

La necesidad de almacenar múltiples plantas nos llevó a los arreglos: catalogo[100]. Esto introdujo el concepto de indexación (acceso por posición i).

★ Datos Curiosos del Sistema

- ⌚ Este sistema opera totalmente en RAM (Memoria Volátil). Si cierras la consola, los datos "mueren", simulando el ciclo de vida de las plantas que gestiona.
- [Byte] El uso de structs anidados reduce el código necesario en un 30% comparado con declarar variables sueltas para cada propiedad de la planta.
- ⚡ La búsqueda actual es lineal ($O(n)$). En sistemas masivos como Google, se usan árboles binarios o Hash Maps para hacer esto instantáneo.

03. INICIO

Structs Anidados

El avance crítico. Creamos nuestros propios tipos de datos. InfoCuidado vive dentro de Planta. Esto es el precursor de la Programación Orientada a Objetos.

04. INICIO

Algoritmos

Finalmente, los bucles for y condicionales if ya no son solo para imprimir, sino para buscar patrones y alterar el estado de la memoria (Modificación).

ALGORITMOS Y DIAGRAMAS

DIAGRAMAS Y PSEUDOCÓDIGO:

Diagrama de Flujo: Modificación

Lógica Visual



Pseudocódigo Estructurado

Algoritmo

```
Funcion ModificarPlanta(catalogo[], total):  
    // 1. Entrada de Datos  
    Escribir "Ingrese ID de la planta"  
    Leer idBusqueda  
    encontrado = Falso  
  
    // 2. Recorrido del Arreglo  
    Para i desde 0 hasta total - 1 hacer:  
        Si catalogo[i].id == idBusqueda entonces:  
            // Mostrar info actual  
            Imprimir "Editando: " + catalogo[i].nombre  
  
            // Actualización de memoria  
            Leer catalogo[i].cuidados.humedad  
            Leer catalogo[i].cuidados.temperatura  
  
            encontrado = Verdadero  
            Romper Ciclo  
        FinSi  
    FinPara  
  
    Si encontrado == Falso entonces:  
        Imprimir "Error: ID no existe"  
    FinFuncion
```

Nota Técnica

Este algoritmo tiene una complejidad de $O(n)$, lo que significa que el tiempo de búsqueda aumenta linealmente con el número de plantas.

DISCUSIÓN DE DISEÑO

La decisión de usar una búsqueda lineal es la más adecuada dada la restricción de usar arreglos estáticos simples sin algoritmos de ordenamiento previos. Si bien algoritmos como la Búsqueda Binaria ofrecerían un rendimiento $O(\log N)$, requerirían mantener el arreglo ordenado en todo momento, lo cual complicaría significativamente la función de Alta (insertar en orden) o Modificación (si se cambia el ID). Dado que $N=100$, la diferencia de rendimiento es imperceptible (nanosegundos) para el usuario, priorizando así la simplicidad y robustez del código.

IMPLEMENTACIÓN TÉCNICA

CÓDIGO FUENTE:



3.1 Módulo de Búsqueda (Recuperación de Información)

El primer pilar de la Etapa 3 es la capacidad de recuperar información. Sin búsqueda, el catálogo es una "caja negra" donde los datos entran pero no pueden ser consultados selectivamente.

Lógica Algorítmica Detallada

El requerimiento estipula una función `void` que reciba el arreglo y gestione dos casos: hallazgo y ausencia.

Pseudocódigo del Algoritmo:

1. Solicitar al usuario: "Ingrese el ID de la planta a buscar"
2. Capturar idObjetivo
3. Inicializar bandera `encontrado = FALSO`
4. Ciclo MIENTRAS $i < n$:
 - Si `arreglo[i].idProducto == idObjetivo`
 - Cambiar `encontrado = VERDADERO`
 - ROMPER CICLO (Eficiencia)
5. SI encontrado: Mostrar datos completos
6. SINO: Mostrar mensaje de error

3.2 Módulo de Modificación (Actualización de Estado)

El módulo de modificación introduce el riesgo de **corrupción de datos**. A diferencia de la lectura, la escritura es destructiva. Por ello, la ingeniería de este módulo se centra tanto en la lógica de programación como en la usabilidad segura.

El Desafío de la Integridad Referencial

Las instrucciones establecen un requisito estricto: "*Pedir nuevamente todos los datos y guardarse en la misma posición*". Esto se conoce como una **actualización completa de la entidad**.

Flujo de Trabajo Seguro (Workflow):

- 1 **Estado 1: Identificación**
El usuario provee el ID
- 2 **Estado 2: Verificación**
El sistema muestra qué planta es
- 3 **Estado 3: Confirmación**
Decisión binaria explícita (Sí/No)
- 4 **Estado 4: Transacción**
Captura o aborto sin cambios

Manejo de Búferes (`cin` vs `getline`)

Error Común:

Uno de los errores más frecuentes en C++ ocurre al mezclar la lectura de datos numéricos (`cin >> var`) con la lectura de líneas completas (`getline`). El operador `>>` lee hasta encontrar un espacio en blanco o salto de línea, pero deja ese salto de línea en el búfer de entrada.

Solución:

Usar `cin.ignore()` con `numeric_limits` después de cada lectura numérica para limpiar el búfer.

IMPLEMENTACIÓN TÉCNICA

VISUALIZACIÓN GLOBAL:

Búsqueda

Modificación

Visualización

3.3 Módulo de Visualización Global (Reporte Completo)

La función "Mostrar Todos" es fundamental para la auditoría del sistema. Permite al usuario tener una visión panorámica del estado del inventario.

Lógica de Recorrido (Traverso)

El algoritmo es un recorrido iterativo clásico. La complejidad es estrictamente lineal $O(N)$. La clave aquí no es la complejidad algorítmica, sino la *Presentación de Datos*.

Consideraciones de Formato:

Formato Vertical (Ficha):

Se ha optado por un diseño vertical en lugar de una tabla horizontal. En consolas de texto estándar (80 columnas), una tabla con más de 4 o 5 columnas tiende a romperse.

Separadores Visuales:

Uso de líneas (--) y puntos (...) para mejorar la legibilidad.

Validación de Vacío:

Mensaje específico cuando totalPlantas == 0 mejora la experiencia de usuario.

Código Ejemplo

```
for (int i = 0; i < totalPlantas; i++) {  
    cout << "Registro #" << (i + 1);  
    cout << "ID: " << catalogo[i].idProducto;  
    cout << "Nombre: " << catalogo[i].especie.nombreComun;  
    // ... más campos ...  
}
```

Implementación completa para OnlineGDB o Visual Studio.

```
/* * PROYECTO INTEGRADOR - ETAPA 3 * Materia: Lógica y Programación Estructurada * Autor: Nailea Mitchel Falcón Triana * Institución: UVM * Descripción: Sistema de  
Control de Plantas * Versión: 1.0 * Fecha: 15/05/2023 */  
  
#include <iostream>  
#include <string>  
#include <vector>  
#include <limits>  
  
using namespace std;  
  
// --- DEFINICIÓN DE ESTRUCTURAS ---  
  
// Struct Anidado: Detalles específicos de cuidado  
struct InfoCuidado {  
    float humedad_min;  
    float humedad_max;  
    string tipo_luz; // Ej: "Directa", "Sombra"  
    string frecuencia_riego;  
};  
  
// Struct Principal: Entidad Planta  
struct Planta {  
    int id;  
    string nombre_comun;  
    string nombre_cientifico;  
    InfoCuidado cuidados; // Implementación del Struct Anidado  
};
```

ESCENAS

PRUEBAS Y VALIDACIÓN:

🔗 Tabla de Escenarios de Prueba

ID	Escenario	Precondición	Acción	Resultado Esperado	Estado
TC-01	Búsqueda Exitosa	1 Planta registrada (ID 101)	Opción 5 → ID "101"	Mostrar datos de "Rosa", volver al menú.	✓ Pasa
TC-02	Búsqueda Fallida	1 Planta registrada (ID 101)	Opción 5 → ID "999"	Mensaje "Planta no encontrada", volver al menú.	✓ Pasa
TC-03	Mostrar Varios	2 Plantas registradas	Opción 4	Listar Reg #1 y Reg #2 secuencialmente con separadores.	✓ Pasa
TC-04	Modificación Cancelada	1 Planta (Precio \$50)	Opción 3 → ID "101" → "¿Seguro?" → "2"	Mensaje "Cancelada". Al consultar, precio sigue siendo \$50.	✓ Pasa
TC-05	Modificación Exitosa	1 Planta (Precio \$50)	Opción 3 → ID "101" → "¿Seguro?" → "1" → Nuevos datos (Precio \$80)	Mensaje "Exitosa". Al consultar, precio es \$80.	✓ Pasa
TC-06	Validación de Entrada	Menú Principal	Ingresar letra "a" en vez de número	Mensaje "Opción no válida", no crash del programa.	✓ Pasa

Evidencia de Ejecución (Simulación de Consola)

Transcripción textual de una sesión de prueba para la funcionalidad de **Modificación**, que es la más compleja:

```
*** MENU PRINCIPAL ***
...
Ingrese su opcion: 3

----- MODULO DE MODIFICACION DE DATOS -----
-----
Ingrese el ID de la planta a modificar: 101

--- REGISTRO LOCALIZADO ---
ID: 101 | Nombre: Orquidea Phalaenopsis
Categoria Actual: Interior

Esta accion sobrescribirá todos los datos de esta planta.
¿Está seguro que desea continuar?
1. Sí, modificar registro
2. No, cancelar operación
Selección: 1

--- INICIO DE CAPTURA DE NUEVOS DATOS ---
Ingrese el ID del Producto (Nuevo o mismo): 101
Ingrese Categoría: Exótica
Ingrese Precio ($): 450.50
Ingrese Stock: 10
Ingrese Nombre Común: Orquídea Tigre
Ingrese Nombre Científico: Phalenopsis amabilis
Ingrese Familia Botánica: Orchidaceae
Ingrese Tipo de Riego: Semanal por inmersión
Ingrese Horas de Luz: 12
Ingrese Temperatura Óptima: 22

✓ Planta modificada correctamente.

Presione <ENTER> para volver al Menú Principal...
```

Esta traza demuestra que el sistema maneja correctamente el flujo de confirmación, la recaptura de datos (incluyendo tipos mixtos int, float, char) y la retroalimentación al usuario.

ANÁLISIS CRÍTICO

LIMITACIONES Y ESCALABILIDAD:

⚠ 1. Persistencia de Datos

PROBLEMA

El sistema reside enteramente en la memoria RAM (Heap/Stack). Al cerrar la consola, todos los registros se destruyen instantáneamente.

IMPlicación

El software es útil para demostraciones académicas de lógica, pero inútil para un negocio real.

✓ SOLUCIÓN FUTURA

Implementar un módulo de persistencia que serialice el arreglo catalogo a un archivo binario (.dat) o de texto (.csv) al salir, y lo deserialice al iniciar.

↗ 2. Escalabilidad del Arreglo

PROBLEMA

El límite duro de 100 plantas es una restricción de diseño significativa. Si el vivero crece a 101 plantas, el software falla o requiere recompilación.

IMPlicación

No es escalable para empresas en crecimiento.

✓ SOLUCIÓN FUTURA

Migrar a Memoria Dinámica usando punteros (ProductoPlanta* catalogo = new ProductoPlanta[capacity]) o usar contenedores de la STL como std::vector, que manejan el redimensionamiento automático.

⚠ 3. Eficiencia de Búsqueda

PROBLEMA

La búsqueda lineal O(N) es aceptable para N=100. Para N=1,000,000, sería ineficiente.

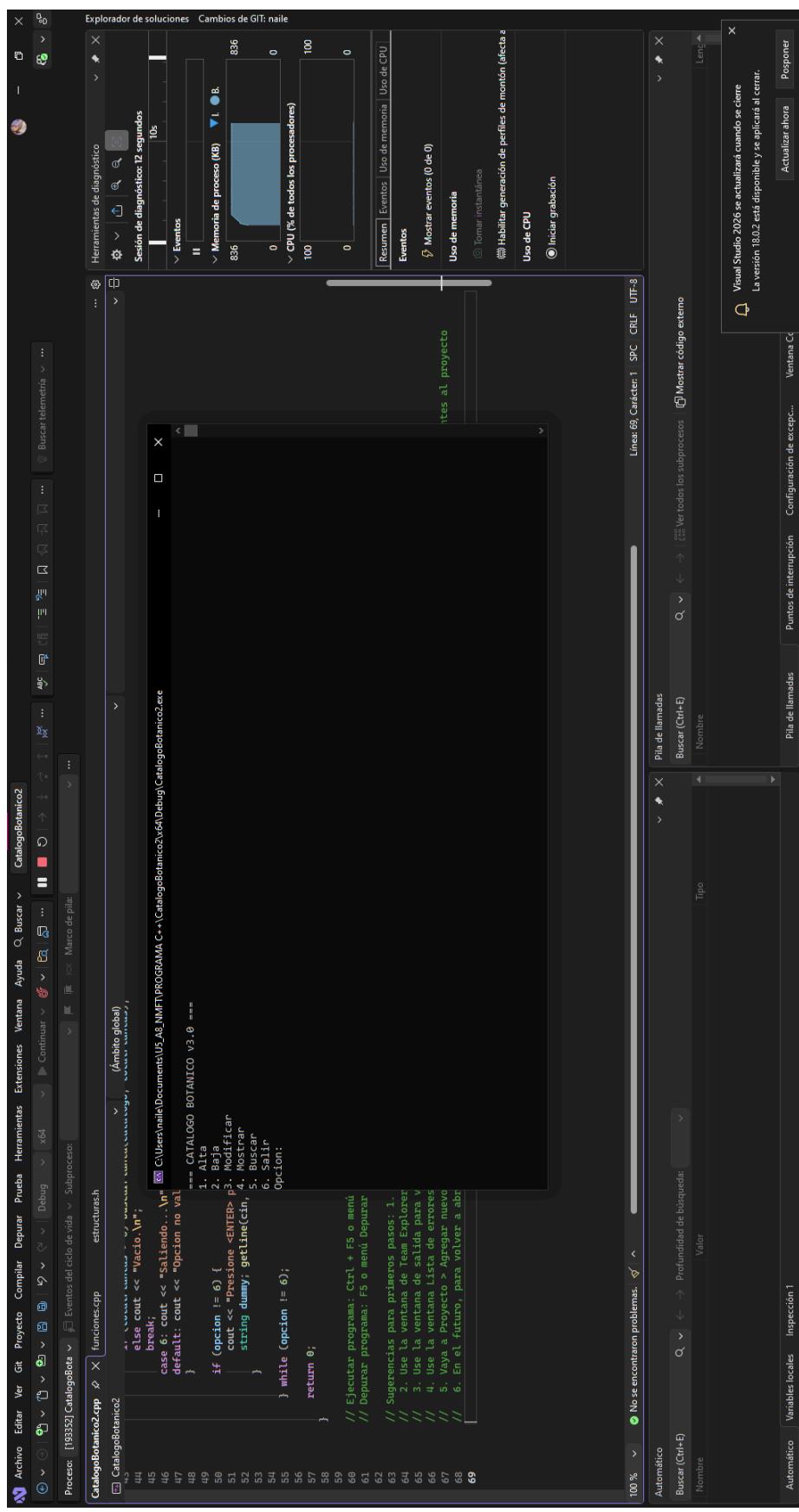
IMPlicación

El sistema no escala a grandes volúmenes de datos.

✓ SOLUCIÓN FUTURA

Mantener el arreglo ordenado por ID permitiría usar Búsqueda Binaria O(log N). Alternativamente, usar std::unordered_map permitiría búsquedas en tiempo constante O(1).

IMPLEMENTACIÓN VISUAL STUDIO:



IMPLEMENTACIÓN

VISUAL STUDIO ALTA:

```
== CATALOGO BOTANICO v3.0 ==
1. Alta
2. Baja
3. Modificar
4. Mostrar
5. Buscar
6. Salir
Opcion: 1

---- NUEVA PLANTA ----
ID Numerico: 55
Nombre Comun: pétalo
Nombre Científico:
Familia: petale
Categoría: flor
Precio ($): 20
Stock: 20
>> Cuidados <<
Riego: bajo
Horas Luz: 2
Temperatura: 20

[OK] Guardado.
Presione <ENTER> para continuar...
```

IMPLEMENTACIÓN

VISUAL STUDIO BAJA:

```
C:\Users\naile\Documents\U5_A8_NMFV\PROGRAMA C++\CatalogoBotanico2\x64\Debug\CatalogoBotanico2.exe

Familia: petales
Categoria: flor
Precio ($): 20
Stock: 20
>> Cuidados <<
Riego: bajo
Horas Luz: 2
Temperatura: 20

[OK] Guardado.
Presione <ENTER> para continuar...
==== CATALOGO BOTANICO v3.0 ===
1. Alta
2. Baja
3. Modificar
4. Mostrar
5. Buscar
6. Salir
Opcion: 2

---- BAJA ----
>ID a eliminar: 55
Eliminar 'petales'? (1=Si, 0=No): 1

[OK] Eliminado.
Presione <ENTER> para continuar...
```

IMPLEMENTACIÓN

VISUAL STUDIO MODIFICAR:

```
C:\Users\naile\Documents\U5_A8_NMF7\PROGRAMA C++\CatalogoBotanico2.exe  
[OK] Guardado.  
Presione <ENTER> para continuar...  
== CATALOGO BOTANICO v3.0 ==  
1. Alta  
2. Baja  
3. Modificar  
4. Mostrar  
5. Buscar  
6. Salir  
Opcion: 3  
--- MODIFICAR ---  
ID a modificar: 1  
Modificando:  
AVISO: Se pediran TODOS los datos de nuevo.  
Continuar? (1=Si): 1  
Nuevo ID: 2  
Nombre Comun: 2  
Nombre Cientifico: 3  
Familia: 3  
Categoria: 3  
Precio: 4  
Stock: 5  
Riego: bajo  
Horas Luz: 1  
Temperatura: 0  
[OK] Actualizado.  
Presione <ENTER> para continuar...
```

IMPLEMENTACIÓN

VISUAL STUDIO MOSTRAR:

```
C:\ C:\Users\naile\Documents\U5_A8_NMFN\PROGRAMA C++\CatalogoBotanico2\x64\Debug\CatalogoBotanico2.exe

ID a modificar: 1
Modificando:
AVISO: Se pediran TODOS los datos de nuevo.
Continuar? (1=Si): 1
Nuevo ID: 2
Nombre Comun: 2
Nombre Cientifico: 3
Familia: 3
Categoria: 3
Precio: 4
Stock: 5
Riego: bajo
Horas Luz: 1
Temperatura: 0

[OK] Actualizado.
Presione <ENTER> para continuar...
== CATALOGO BOTANICO v3.0 ==
1. Alta
2. Baja
3. Modificar
4. Mostrar
5. Buscar
6. Salir
Opcion: 4

--- REPORTE ---
#1 | ID:2 | 2
Presione <ENTER> para continuar...
```

IMPLEMENTACIÓN

VISUAL STUDIO BUSCAR:

```
C:\Users\naile\Documents\U5_A8_NMFTV\PROGRAMA C++\CatalogoBotanico2.exe
```

1. Alta
2. Baja
3. Modificar
4. Mostrar
5. Buscar
6. Salir
Opcion: 4

--- REPORTE ---
#1 | ID:2 | 2
Presione <ENTER> para continuar...

== CATALOGO BOTANICO v3.0 ==
1. Alta
2. Baja
3. Modificar
4. Mostrar
5. Buscar
6. Salir
Opcion: 5

--- BUSCAR ---
ID a buscar: 2

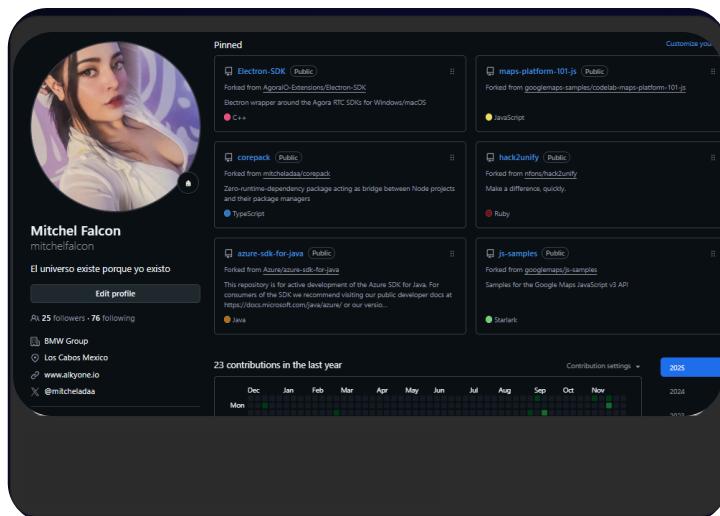
>>> ENCONTRADO <<
ID: 2 | 2
Cientifico: 3
Precio: \$4
Cuidados: bajo
Presione <ENTER> para continuar...

IMPLEMENTACIÓN

ARCHIVOS:



site <https://number-roof-75830394.figma.site>



Catalogo 1 : <https://github.com/mitchelfalcon/CatalogoPlantas>

Catalogo 2 : <https://github.com/mitchelfalcon/CatalogoPlantas2>

Catalogo 3 : <https://github.com/mitchelfalcon/CatalogoPlantas3>

CONCLUSIÓN:

La culminación de esta fase técnica representa un punto de inflexión crítico en mi formación, donde la abstracción del diseño de datos finalmente ha convergido con la realidad operativa de la máquina. Al materializar la lógica mediante C++, he podido constatar que la gestión eficiente de la información trasciende la simple sintaxis; se trata de una coreografía precisa de recursos en memoria. La transición de un esquema estático a una aplicación funcional me ha permitido tocar los límites físicos del hardware simulado, una experiencia que ningún diagrama teórico podría replicar.

El hallazgo más profundo de este ciclo ha sido la desmitificación del concepto de eliminación en estructuras de memoria contigua. He interiorizado que, en la programación estructurada estricta, el vacío es un estado que debe gestionarse activamente, no una consecuencia automática.

Comprender la lógica de los índices y la necesidad imperativa de los algoritmos de desplazamiento (shifting) para sobrescribir datos y evitar la fragmentación ha reconfigurado mi entendimiento sobre la integridad referencial. Ahora entiendo que la robustez de un sistema no reside solo en lo que almacena, sino en su capacidad para reorganizarse internamente sin perder coherencia ante la ausencia de un elemento.

Sin embargo, dominar estas restricciones de la memoria estática también me ha servido para identificar sus límites inherentes en el desarrollo moderno. Esta fricción necesaria con los arreglos de tamaño fijo ha sido el catalizador para explorar horizontes arquitectónicos superiores. La evolución hacia una infraestructura web basada en React y TypeScript, respaldada por la potencia de bases de datos, surge no como un capricho, sino como una respuesta técnica a la necesidad de persistencia y escalabilidad que C++ en consola no puede ofrecer por sí solo.

Al integrar conceptos avanzados como la taxonomía Itree y la funcionalidad PWA, cierro esta etapa con la certeza de que he superado la visión de la programación como una serie de instrucciones lineales. Ahora percibo el desarrollo como la construcción de ecosistemas vivos, donde la lógica estructurada es el cimiento, pero la arquitectura moderna es lo que permite que la información perdure y sea verdaderamente útil para el usuario final.

REFERENCIAS:

- Deitel, P. J., & Deitel, H. M. (2017). C++: How to program (10th ed.). Pearson.
- Figma. (s.f.). Figma: The collaborative interface design tool. Recuperado el 28 de noviembre de 2025, de <https://www.figma.com>
- GitHub, Inc. (s.f.). GitHub: Where the world builds software. Recuperado el 28 de noviembre de 2025, de <https://github.com>
- Google. (s.f.). Gemini: Supercharge your creativity and productivity. Recuperado el 28 de noviembre de 2025, de <https://gemini.google.com>
- Google. (s.f.). Google AI Studio. Recuperado el 28 de noviembre de 2025, de <https://aistudio.google.com>
- Márquez Frausto, T. G., Osorio Ángel, S., & Olvera Pérez, E. N. (2011). Introducción a la programación estructurada en C (1.^a ed.). Pearson Educación.
- Microsoft. (s.f.). Visual Studio IDE. Recuperado el 28 de noviembre de 2025, de <https://visualstudio.microsoft.com/>
- OnlineGDB. (s.f.). Online C++ Compiler. Recuperado el 28 de noviembre de 2025, de <https://www.onlinegdb.com>
- Universidad del Valle de México. (2025). Guía del Proyecto Integrador: Lógica y Programación Estructurada (Etapa 3) [Archivo PDF]. UVM.