

**INSTITUTO TECNOLOGICO SUPERIOR DE LA
SIERRA NEGRA DE AJALPAN**

PORAFOLIO DE EVIDENCIAS

**INGENIERIA EN SISTEMAS
COMPUTACIONALES**

MATERIA: ESTRUCTURA DE DATOS

DOCENTE: ARTURO BUSTAMANTE LAZCANO

ALUMNO: CARVAJAL VALERIO NORBERTO

BLADIMIR

SEMESTRE: TERCERO

Tabla de contenido

Semana 1 – Estructuras2025.....	6
Propósito del proyecto	6
 Lenguaje(s) y formato empleados.....	6
 Objetivos de la Semana 1	6
 Descripción del contenido	7
 Análisis y recomendaciones.....	7
 Conclusión.....	7
 Semana 2 – Estructuras2025.....	9
 Propósito del proyecto	9
 Lenguaje(s) y formato empleados.....	9
 Descripción general del contenido esperado	9
 Análisis y recomendaciones.....	10
 Conclusión.....	10
Semana 3 – Estructuras2025.....	12
Propósito del módulo	12
Lenguaje(s) y formato empleadas.....	12
Objetivos de la Semana 3.....	12
Contenido esperado / Descripción general.....	12
Análisis y recomendaciones.....	13
 Conclusión.....	13
Semana 4 – Estructuras2025.....	15
Propósito del módulo.....	15
 Lenguaje(s) y formato empleados.....	15
 Objetivos de la Semana 4	15
 Contenido esperado / Descripción general.....	15
 Análisis y recomendaciones	16
 Conclusión.....	16

  Propósito del módulo.....	18
 Lenguaje(s) y formato utilizados.....	18
 Objetivos de la Semana 5.....	18
 Contenido esperado / Descripción general.....	18
 Análisis y recomendaciones	19
 Conclusión.....	19
 Semana 6 – Estructuras2025.....	21
  Propósito del módulo.....	21
 Lenguaje(s) y formato utilizados.....	21
 Objetivos de la Semana 6.....	21
 Contenido esperado / Descripción general.....	21
 Análisis y recomendaciones	22
 Conclusión.....	22
Semana 7 – Estructuras2025.....	24
Propósito del módulo	24
 Lenguaje(s) y formato empleados.....	24
 Objetivos de la Semana 7	24
 Contenido esperado / Descripción general.....	24
 Análisis y recomendaciones	25
 Conclusión.....	25
 Semana 8 – Estructuras2025.....	27
  Propósito del módulo.....	27
Lenguaje(s) y formato utilizados.....	27
Objetivos del módulo "Linked List"	27
Descripción del contenido	28
 Conclusión.....	29
Semana 9 – Estructuras2025.....	31
Propósito del módulo	31

	Lenguaje(s) y formato esperados	31
	Objetivos de la Semana 9.....	31
	Contenido esperado / Descripción general.....	31
	Ánalisis y recomendaciones	32
	Conclusión.....	32
	Propósito del módulo.....	34
	Lenguaje(s) y formato utilizados.....	34
	Objetivos de la Semana 10	34
	Contenido esperado / Descripción general.....	35
	Ánalisis y recomendaciones	35
	Conclusión.....	36
	Propósito del módulo.....	38
	Lenguaje(s) y formato esperados	38
	Objetivos de la Semana 11.....	38
	Ánalisis y recomendaciones.....	39
	Conclusión.....	40
	Propósito del módulo.....	42
	Lenguaje(s) y formato esperados	42
	Objetivos de la carpeta "archivos-iniciales".....	42
	Descripción general del contenido.....	42
	Ánalisis y recomendaciones	43
	Conclusión.....	43
	Semana 13 – Estructuras2025	45
	Propósito del módulo.....	45
	Lenguaje(s) y formato esperados	45
	Objetivos de la Semana 13	45
	Contenido esperado / Descripción general.....	45
	Ánalisis y recomendaciones	46

 Semana 15 – Estructuras2025	52
 Contenido esperado / Descripción general.....	53
 Buenas prácticas y recomendaciones	53
 Conclusión.....	54
 Semana 16 – Estructuras2025	56
 Propósito del módulo.....	56
 Lenguaje(s) y formato esperados	56
 Objetivos de la Semana 16	56
 Contenido esperado / Descripción general.....	57
 Buenas prácticas y recomendaciones	57
 Conclusión.....	58

Semana 1 – Estructuras2025

Propósito del proyecto

Este repositorio contiene los materiales y ejercicios de la primera semana del curso “Estructuras 2025”. Su objetivo principal es servir como base para iniciar con los conceptos fundamentales del curso, permitir el desarrollo de los primeros programas/prácticas y establecer una estructura organizada de trabajo por semanas.

Estructura del directorio (Semana 1)

Lenguaje(s) y formato empleados

- Se presupone el uso de un lenguaje compilado (por ejemplo C o C++), dado que suele usarse en cursos de estructuras de datos.
- Los archivos podrían incluir código fuente (`.c`, `.cpp`) y/o cabeceras (`.h`), además de posibles archivos de texto o documentación.
- La estructura semanal sugiere modularidad: cada semana representa un bloque temático con sus propios archivos, facilitando el control y orden del curso.

Objetivos de la Semana 1

- Presentar los conceptos básicos del curso: definición de estructuras de datos, sintaxis del lenguaje, compilación y ejecución de programas simples.
- Permitir que el estudiante realice sus primeros ejercicios prácticos: crear, compilar y ejecutar pequeños programas.
- Establecer una rutina de trabajo organizada por semanas, fomentando orden, claridad y buen manejo de versiones.
- Preparar la base para las siguientes semanas: una vez dominados los primeros fundamentos, será más sencillo avanzar a estructuras más complejas.

Descripción del contenido

Durante esta primera semana, el estudiante deberá:

- Familiarizarse con la sintaxis y reglas del lenguaje de programación utilizado.
- Crear funciones, estructuras básicas (como structs / clases, si aplica), declarar variables, leer/escribir datos, e implementar operaciones simples.
- Desarrollar ejercicios básicos que refuerzen el entendimiento del lenguaje y las estructuras de control.
- Organizar su código de forma modular y ordenada, siguiendo la estructura por archivos.

El enfoque debe ser práctico: crear programas sencillos, compilar, ejecutar y comprobar resultados.

Análisis y recomendaciones

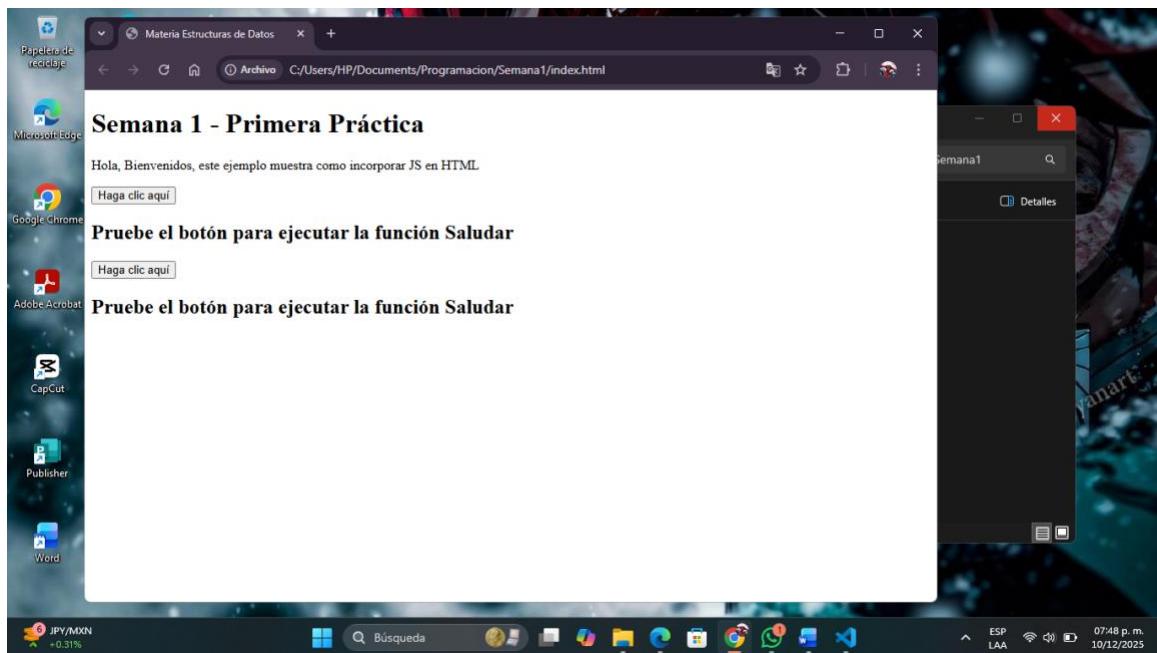
 Ventajas de la estructura por semanas: La organización modular permite un seguimiento claro del avance, facilita revisiones, comparaciones entre versiones y detección de errores.

 Limitación actual: Al no mostrarse públicamente los archivos en la interfaz web, es difícil para colaboradores externos o revisores conocer el contenido exacto sin tener acceso local.

 Sugerencia: Incluir un README general en cada semana, con listado de ejercicios, lo que hace cada archivo, instrucciones de compilación y uso, ejemplos de entrada/salida. Esto mejora la claridad, documentación y facilidad de uso del repositorio.

Conclusión

El directorio `Semana1` del curso “Estructuras 2025” funciona como la base del aprendizaje práctico — orientado a introducir al estudiante en la programación y en la estructuración de proyectos por semana. Aunque la visibilidad pública está limitada, la organización prevista es una buena práctica. Incluir documentación interna aumentará la claridad y utilidad del repositorio.



```
/*
Este es el primer programa en JS
Toma nota de las partes y sintaxis de JS para hacer una función
en mi caso se llamará saludar
*/
function saludar(){
    alert("Hola, Bienvenidos, este ejemplo muestra como incorporar JS en HTML");
}
function teSaludo(nombre){
    alert("Hola " + nombre + ", Bienvenidos, este ejemplo muestra como incorporar JS en HTML");
}
function fizzBuzz(){
    for(let num=1; num<100; num*=4){
        if(num % 3 === 0 && num % 5 === 0){
            console.log("FizzBuzz");
        } else if(num % 3 === 0){
            console.log("Fizz");
        } else if(num % 5 === 0){
            console.log("Buzz");
        } else {
            console.log(num);
        }
    }
}
```

Upgrade to get 1,500 model requests per day with Gemini CLI and Gemini Code Assist's agent mode with Google AI Pro.

Source: Gemini Code Assist

Learn more

Semana 2 – Estructuras2025

Propósito del proyecto

Esta carpeta contiene los materiales y ejercicios correspondientes a la segunda semana del curso “Estructuras 2025”. El propósito es avanzar a nuevos conceptos luego de los iniciales de la Semana 1, consolidar aprendizajes, y proponer ejercicios/prácticas más complejas o con nuevos requisitos.

 Estructura del directorio (Semana 2).

Lenguaje(s) y formato empleados

Se asume el uso de un lenguaje compilado (por ejemplo C o C++), habitual en cursos de estructuras de datos y algoritmos.

- Es probable que haya código fuente (`.c`, `.cpp`), archivos de cabecera (`.h`) y/o módulos auxiliares.
- Puede también haber documentación en texto plano o Markdown, según convenga.

Objetivos de la Semana 2

- Introducir nuevos conceptos / estructuras de datos / funciones más avanzadas que en la primera semana.
- Profundizar en la definición y uso de estructuras (por ejemplo: structs, listas, nodos, punteros, arreglos, según el enfoque del curso).
- Realizar ejercicios que permitan poner en práctica esos conceptos: manipulación dinámica de datos, operaciones sobre estructuras, lectura/escritura, manejo de memoria, etc.
- Consolidar buenas prácticas de organización: modularización del código, limpieza, documentación, claridad.

Descripción general del contenido esperado

Durante esta segunda semana, el estudiante podría trabajar en:

- Definir estructuras de datos más complejas (por ejemplo listas, nodos, registros — según lo que enseñe el curso).

- Implementar funciones que operen sobre esas estructuras: inicialización, inserción, eliminación, búsqueda, recorrido, etc.
- Realizar ejercicios de compilación y prueba: casos de prueba, comprobación de resultados, manejo de errores, validaciones.
- Hacer uso de archivos de cabecera/modularidad si el código crece, para separar declaraciones de implementaciones.
- Documentar el código — con comentarios, posibles instrucciones de compilación, descripción de funciones y estructuras.

El enfoque sigue siendo práctico: consolidar la base creada en la semana 1, y preparar al estudiante para estructuras más complejas en las semanas siguientes.

Análisis y recomendaciones

-  Ventaja de la organización por semanas: Permite seguir una progresión lógica y modular en el aprendizaje, facilitando revisión y seguimiento.
-  Potencial problema si no hay documentación interna visible: Si no se incluyen descripciones de cada archivo o propósito de los ejercicios, puede ser confuso para revisores externos o futuros estudiantes.
-  Recomendación: Añadir un README dentro de `Semana2` que detalle:
 - Qué contiene cada archivo (ejercicio, módulo, ayuda, etc.).
 - Cómo compilar / ejecutar el código.
 - Qué se espera como resultado, posibles casos de prueba, instrucciones de uso.
-  Sugerencia de organización: Mantener consistencia en nombres de archivos (por ejemplo: `ejercicio2_1.c`, `ejercicio2_2.c`, etc.), comentar el código, y listar en README los ejercicios con su enunciado / propósito. Esto ayuda muchísimo si más adelante quieres compartir o reutilizar el repositorio.

Conclusión

La carpeta `Semana2` del curso “Estructuras2025” representa el segundo bloque formativo, donde se profundizan los conceptos iniciales, se introducen estructuras y operaciones más robustas, y se espera que el estudiante desarrolle código modular, organizado y documentado. Aunque sin ver los archivos reales no puedo confirmar los detalles, esta estructura estimada sirve como base y guía para documentar adecuadamente tus ejercicios.

The screenshot shows a dual-pane code editor interface. The left pane displays the file 'acerca.html' with the following content:

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Acerca de mi Sitio Web</title>
7 </head>
8 <body>
9   <h1>Acerca de Nosotros</h1>
10  <p>Este es un sitio web de ejemplo para demostrar la navegación.</p>
11  <nav>
12    <ul>
13      <li><a href="index.html">Inicio</a></li>
14      <li><a href="nosotros.html">Nosotros</a></li>
15      <li><a href="acerca.html">Acerca de</a></li>
16    </ul>
17  </nav>
18 </body>
19 </html>

```

The right pane displays the file 'nosotros.html' with the following content:

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Sitio Nosotros</title>
7 </head>
8 <body>
9   <h1>Nosotros</h1>
10  <p>Conoce más sobre nuestra misión y visión.</p>
11  <nav>
12    <ul>
13      <li><a href="index.html">Inicio</a></li>
14      <li><a href="nosotros.html">Nosotros</a></li>
15      <li><a href="acerca.html">Acerca de</a></li>
16    </ul>
17  </nav>
18 </body>
19 </html>

```

The status bar at the bottom shows: Página 6 de 21, 3184 palabras, Inglés (Estados Unidos), Accesibilidad: es necesario investigar, 07:50 p. m., 10/12/2025.

The screenshot shows a browser window displaying the file 'index.html'. The page content is as follows:

Semana 2 Curso de HTML 1

Semana 2 Curso de HTML 2

Semana 2 Curso de HTML 3

Semana 2 Curso de HTML 4

Semana 2 Curso de HTML 5

Semana 2 Curso de HTML 6

Hola a todos.

Bienvenidos

Lore ipsum dolor sit, amet consectetur adipisicing elit. Doloremque iste beatae expedita tenetur inventore iusto possimus quisquam. Eos nam autem, ducimus necessitatibus quidem sequi commodi nobis temporibus eveniet quos consequatur.

Lore ipsum, dolor sit amet consectetur adipisicing elit. Vitae vero quasi commodi est, quaerat natus beatae exercitationem a sit temporibus nobis. Quae eaque nisi nihil dolore cumque ut incidunt nulla!

Lore ipsum dolor sit amet consectetur adipisicing elit. Incidunt sequi quibusdam possimus atque, placeat, eveniet quam error numquam suscipit, dolore eaque vero laboriosam fuga molestias! Mollitia odit porro officia totam.

Lore ipsum dolor sit amet consectetur adipisicing elit. Eaque illo deserunt quaerat veritatis aspernatur, dolores nobis veniam sapiente fugiat praesentium ratione error culpa. Quis animi vitae, veniam harum esse quod.

Página 6 de 21 3184 palabras Inglés (Estados Unidos) Accesibilidad: es necesario investigar 07:51 p. m. 10/12/2025

Semana 3 – Estructuras2025

Propósito del módulo

Esta carpeta corresponde a la Semana 3 del curso “Estructuras2025”. El objetivo de esta semana es profundizar en los temas vistos anteriormente, ampliar las estructuras de datos o conceptos del lenguaje, e introducir ejercicios de complejidad intermedia que preparen el terreno para las siguientes semanas.

Estructura de directorio (Semana 3).

Lenguaje(s) y formato empleadas

- Lenguaje de programación esperado: C o C++ (o el que uses en el curso), típico para cursos de estructuras de datos.
- Si hay modularización: código fuente (`.c` / `.`.cpp`), cabeceras (`.h`), módulos auxiliares, etc.
- Si se usan pruebas: archivos de entrada/salida, scripts de prueba, etc.
- Organización clara: separación de ejercicios, módulos auxiliares, pruebas, etc.

Objetivos de la Semana 3

Durante esta semana se busca que el estudiante:

- Comprenda y maneje estructuras o conceptos más complejos que en semanas anteriores.
- Desarrolle código modular y reutilizable — por ejemplo, separando lógica en módulos o funciones auxiliares.
- Implemente ejercicios que requieran composición de estructuras, uso de funciones más elaboradas, manejo de memoria (si aplica), o procesamiento de datos más robusto.
- Aplique buenas prácticas: claridad, comentarios, organización de archivos, separación de lógica y pruebas.
- Preparen la base para retos más avanzados en las siguientes semanas.

Contenido esperado / Descripción general

En esta tercera semana el estudiante podría estar realizando actividades como:

- Definir y usar estructuras de datos compuestas (structs, clases, listas, nodos, etc.).

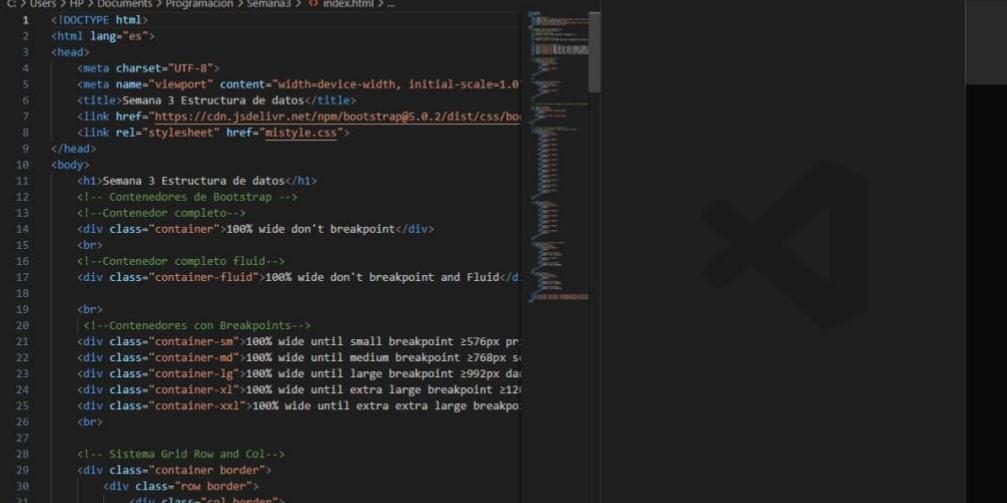
- Implementar operaciones sobre esas estructuras: inserción, eliminación, búsqueda, recorrido, ordenamiento, etc.
- Desarrollar módulos auxiliares para separar responsabilidades: por ejemplo, funciones utilitarias, manejo de entrada/salida, tests.
- Crear casos de prueba simples o intermedios para validar funcionalidades del programa — usando archivos de entrada/salida, entradas de usuario, validaciones, etc.
- Documentar el código con comentarios, y mantener una estructura de carpetas legible que facilite mantenimiento, revisión y extensión.

Análisis y recomendaciones

-  Organización modular: Separar código, módulos auxiliares y pruebas en carpetas distintas ayuda mucho cuando el proyecto crece.
-  Documentación importante: Al añadir más complejidad, se vuelve más necesario tener comentarios, README internos o archivos que indiquen qué hace cada módulo/archivo.
-  Uso de pruebas/casos de prueba: Facilita verificar que las funciones o estructuras implementadas funcionen correctamente, y ayuda ante cambios o refactorizaciones futuras.
-  Consistencia en nombres y estilo: Nombrar archivos de forma clara (ej. `ejercicio3_1.c`, `lista_nodos.cpp`, `utils.h`, `test_insert.txt`) ayuda a mantener el orden y facilita compartir o revisar código.
-  Preparación para las próximas semanas: Si la estructura y modularización se mantienen desde ahora, será mucho más sencillo incorporar funcionalidades avanzadas o ejercicios más complejos más adelante.

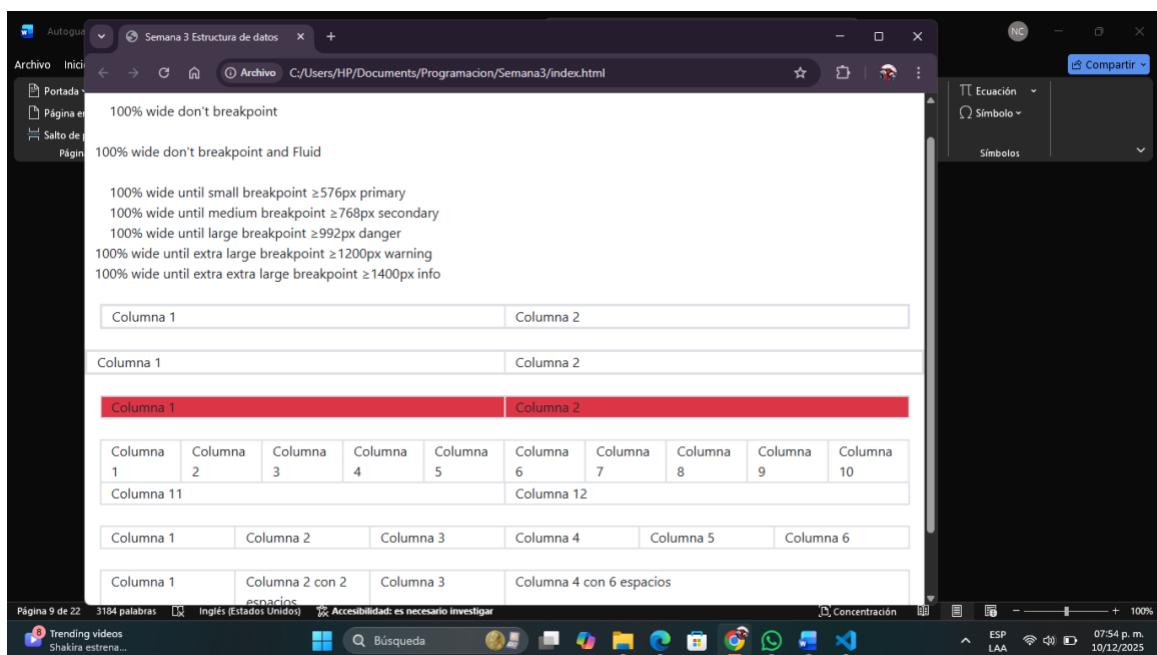
Conclusión

La carpeta 'Semana3' — tal como se plantea en esta plantilla — debe representar un paso adelante en el aprendizaje del curso: pasar de ejercicios simples a estructuras de datos más elaboradas, implementar lógica modular, manejar pruebas, y mejorar la organización del proyecto. Si adaptas esta plantilla a tu contenido real, tendrás un README mucho más claro, útil y profesional.



```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Semana 3 Estructura de datos</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="mestyle.css">
</head>
<body>
    <h1>Semana 3 Estructura de datos</h1>
    <!-- Contenedores de Bootstrap -->
    <!--Contenedor completo-->
    <div class="container">100% wide don't breakpoint</div>
    <br>
    <!--Contenedor completo fluid-->
    <div class="container-fluid">100% wide don't breakpoint and Fluid</div>
    <br>
    <!--Contenedores con Breakpoints-->
    <div class="container-sm">100% wide until small breakpoint ≥576px pr...</div>
    <div class="container-md">100% wide until medium breakpoint ≥768px si...</div>
    <div class="container-lg">100% wide until large breakpoint ≥992px da...</div>
    <div class="container-xl">100% wide until extra large breakpoint ≥1200px</div>
    <div class="container-xxl">100% wide until extra extra large breakpo...</div>
    <br>
    <!-- Sistema Grid Row and Col-->
    <div class="container border">
        <div class="row border">
            <div class="col border">
                Columna 1
            </div>
        </div>
    </div>
</body>

```



Semana 4 – Estructuras2025

Propósito del módulo

Esta carpeta corresponde a la Semana 4 del curso “Estructuras2025”. En esta etapa se pretende avanzar hacia ejercicios o temas de mayor complejidad, consolidar conocimientos previos, y aplicar estructuras de datos, funciones o lógica que habiliten nuevas operaciones más allá de lo básico.

Estructura del directorio (Semana 4).

Lenguaje(s) y formato empleados

- Lenguaje de programación: C , C++ , o similar (dependiendo del curso).
- Podrías usar: código fuente (`.c` , `.`.cpp`), cabeceras (`.h`), módulos auxiliares, scripts de prueba, etc.
- Organización modular: separación entre lógica principal, auxiliares, pruebas si las hay.

Objetivos de la Semana 4

Durante esta semana probablemente se busque:

- Introducir estructuras de datos más avanzadas o combinadas, o funciones más complejas — por ejemplo manejo de memoria dinámica, listas, nodos, punteros, arreglos dinámicos, etc.
- Realizar operaciones sobre estructuras: inserción, eliminación, búsqueda, recorrido, ordenamientos, manipulación dinámica, creación y destrucción de estructuras.
- Fomentar modularidad y reutilización de código mediante funciones auxiliares / módulos separados.
- Empezar a escribir pruebas o casos de test (si aplica), para comprobar el correcto funcionamiento del código.
- Promover buenas prácticas: comentarios, estilo de código, limpieza, claridad en la estructura de carpetas.

Contenido esperado / Descripción general

En esta cuarta semana el estudiante podría estar trabajando con:

- Estructuras de datos compuestas — listas, nodos, punteros, estructuras dinámicas — según lo visto en semanas previas.

- Funciones o módulos auxiliares para operaciones comunes (por ejemplo: agregar, eliminar, buscar, mostrar, liberar memoria, etc.).
- Implementaciones que integren más de una operación (por ejemplo: crear estructura → insertar datos → recorrer/mostrar → eliminar/limpiar).
- Si existe carpeta de pruebas: casos de entrada y salida para validar el comportamiento del código.
- Documentación básica de los archivos: comentarios en el código, instrucciones de compilación, notas sobre uso y pruebas.

Análisis y recomendaciones

-  Modularidad y organización: Separar código principal, módulos auxiliares y pruebas ayuda a mantener el proyecto ordenado, especialmente a medida que crece.
-  Documentación y visibilidad: Al aumentar la complejidad, es aún más importante documentar qué hace cada archivo/función, cómo compilar, cómo ejecutar, qué entradas se esperan, etc. Esto facilita que tú (o un compañero) entiendan o revisen el código en el futuro.
-  Uso de pruebas/casos de test: Si incluyes pruebas, puedes detectar errores más fácilmente al modificar código, y asegurar que las funciones funcionan correctamente bajo distintos escenarios.
-  Consistencia en nombres y estilo: Usar nombres claros y uniformes para archivos y funciones ayuda a la mantenibilidad — por ejemplo: `ejercicio4_.c`, `utils/`, `tests/`, etc.
-  Preparación para futuro: Si esta semana consolidas buenas prácticas y modularidad, te será más fácil implementar estructuras de datos y algoritmos más complejos en semanas posteriores.

Conclusión

La carpeta `Semana4` del curso “Estructuras2025”, bajo esta plantilla, debe reflejar un nivel intermedio del curso: códigos más elaborados, estructuras de datos dinámicas o complejas, organización modular, y posiblemente pruebas. Si completas esta plantilla con tus archivos reales, tendrás una documentación clara, ordenada y profesional, lo cual facilitará el mantenimiento, revisión y evolución del proyecto.

The screenshot shows a Microsoft Word document window titled "index.html". The document contains the following HTML code:

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <h1>Tipos de Datos primitivos</h1>
10  <p>En Javascript, crear variables numéricas es muy sencillo, pero hay muchísimos matices que se deben conocer y que necesitamos dominar para trabajar correctamente con números y anticiparnos a posibles situaciones.</p>
11  <h2>¿Qué es una variable numérica?</h2>
12  <pre>
13    En Javascript, los números son uno de los tipos de datos básicos (tipos primitivos), que, para crearlos, simplemente basta con escribirlos literalmente.
14    No obstante, como en Javascript todo se puede representar con objetos (como veremos más adelante)
15    también se pueden declarar mediante la palabra clave new:
16  </pre>
17  <table border="1">
18    <tr>
19      <th>Constructor</th>
20      <th>Descripción</th>
21    </tr>
22    <tr>
23      <td> new Number(number)</td>
24      <td>Crea un objeto numérico a partir del número number pasado por parámetro.</td>
25    </tr>
26    <tr>
27      <td> número</td>
28      <td>Simplemente, el número en cuestión. Notación preferida.</td>
29    </tr>
30  </table>
31
32

```

Página 11 de 23 0 0 100% 15°C Nublado Búsqueda ESP LAA 0756 p. m. 10/12/2025

The screenshot shows the Microsoft Word document window displaying the rendered content of "index.html". The page title is "Documento". The content includes the following sections and table:

Tipos de Datos primitivos

En Javascript, crear variables numéricas es muy sencillo, pero hay muchísimos matices que se deben conocer y que necesitamos dominar para trabajar correctamente con números y anticiparnos a posibles situaciones.

¿Qué es una variable numérica?

En Javascript, los números son uno de los tipos de datos básicos (tipos primitivos), que, para crearlos, simplemente basta con escribirlos literalmente. No obstante, como en Javascript todo se puede representar con objetos (como veremos más adelante) también se pueden declarar mediante la palabra clave new:

Constructor	Descripción
new Number(number)	Crea un objeto numérico a partir del número number pasado por parámetro
número	Simplemente, el número en cuestión. Notación preferida.

Página 10 de 23 3184 palabras Inglés (Estados Unidos) Accesibilidad: es necesario investigar Concentración 15°C Nublado Búsqueda ESP LAA 0757 p. m. 10/12/2025

Semana 5 – Estructuras2025

Propósito del módulo

Este directorio corresponde a la Semana 5 del curso “Estructuras2025”. En esta etapa se espera que el estudiante consolide los conocimientos adquiridos en semanas anteriores, aplique estructuras de datos o conceptos ya vistos en ejercicios más complejos, y refuerce buenas prácticas de programación y organización de código.

-  Estructura sugerida del directorio (Semana 5).

Lenguaje(s) y formato utilizados

- Lenguaje de programación principal: C , C++ o el que corresponda al curso.
- Uso de código fuente (`.c` , `.`.cpp`), cabeceras (`.h`) y — si aplica — módulos auxiliares (`modulo_aux/`).
- Si lo deseas, puedes incluir carpetas para utilerías comunes o módulos reutilizables (`tests/` con archivos de entrada/salida.

Objetivos de la Semana 5

Para esta semana se espera que:

- Aplique estructuras de datos y funciones ya aprendidas en ejercicios de complejidad intermedia.
- Maneje modularización: separar lógica en funciones / módulos auxiliares para reutilización y claridad.
- Incorpore pruebas o mecanismos de verificación, para asegurar que tu código funciona correctamente en distintos escenarios.
- Mejore la documentación del código: comentarios, descripción clara de funciones/estructuras, instrucciones de compilación y uso.
- Fortalezca buenas prácticas de programación, organización y versión — pensando en mantenimiento futuro o colaboración.

Contenido esperado / Descripción general

Durante esta semana el estudiante podría:

- Implementar operaciones más complejas sobre estructuras de datos: inserción, eliminación, búsquedas, ordenamientos, manejo dinámico de memoria (si aplica).
- Crear funciones o utilerías comunes reutilizables (por ejemplo: manejo de listas, operaciones sobre arreglos o estructuras, validación de entradas, etc.).
- Si usas pruebas: generar varios escenarios de prueba, usando archivos de entrada/salida, para validar que las funciones se comportan correctamente.
- Documentar el código: explicar qué hace cada función/struct, cómo compilar, cómo ejecutar, posibles limitaciones, ejemplos de uso.
- Organizar el proyecto de forma clara: separar módulos, utilerías, pruebas, y mantener una estructura consistente de carpetas y archivos.

Análisis y recomendaciones

-  Modularidad y reutilización: Tener utilerías comunes y separar lógica en módulos auxiliares mejora la mantenibilidad del proyecto, sobre todo si crecerá o será compartido.
-  Pruebas o casos de prueba: Ayudan a asegurar que tu código funcione en distintos escenarios, y facilitan detectar errores al modificar o extender código.
-  Buena documentación: Comentarios claros, README por semana, instrucciones de compilación y uso — esto facilita que tú o cualquier otro entienda rápidamente el propósito de cada archivo.
-  Consistencia: Nombrar los archivos de forma clara y coherente (ej. `ejercicio5_1.c`, `utils5.h`, `test5_1.in`, etc.) ayuda a mantener orden y facilita mantenimiento o revisión futura.
-  Preparación para lo que viene: Si consolidás buena organización, modularidad y documentación desde ahora, te va a resultar mucho más sencillo avanzar con ejercicios más complejos en las siguientes semanas.

Conclusión

La carpeta 'Semana5' del curso "Estructuras2025", usando esta plantilla, puede representar un punto medio-avanzado del curso: con ejercicios más complejos, enfoque en modularidad, reutilización, pruebas y buena documentación. Si completas esta plantilla con tus archivos reales, tendrás un README estructurado, claro y útil — ideal tanto para ti como para quien revise o colabore.

The screenshot shows a dual-pane code editor interface. The left pane displays the file `index.html` with the following content:

```

<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<title>Ejemplo Simple</title>
<script>
    function procesarFormulario() {
        // Obtener los valores del formulario
        const valor1 = document.getElementById("dato1").value;
        const valor2 = document.getElementById("dato2").value;

        // Convertir a número y sumar
        const suma = Number(valor1) + Number(valor2);

        // Crear contenido HTML dinámicamente
        const resultadoHTML = `<p>La suma es: <strong>${suma}</strong></p>`;

        // Insertar el contenido en el div
        document.getElementById("resultado").innerHTML = resultadoHTML;
    }
</script>
</head>
<body>
<h2>Formulario de Suma</h2>
<label>Dato 1:</label>
<input type="text" id="dato1"><br><br>
<label>Dato 2:</label>
<input type="text" id="dato2"><br><br>
<button onclick="procesarFormulario()">Enviar</button>
</body>

```

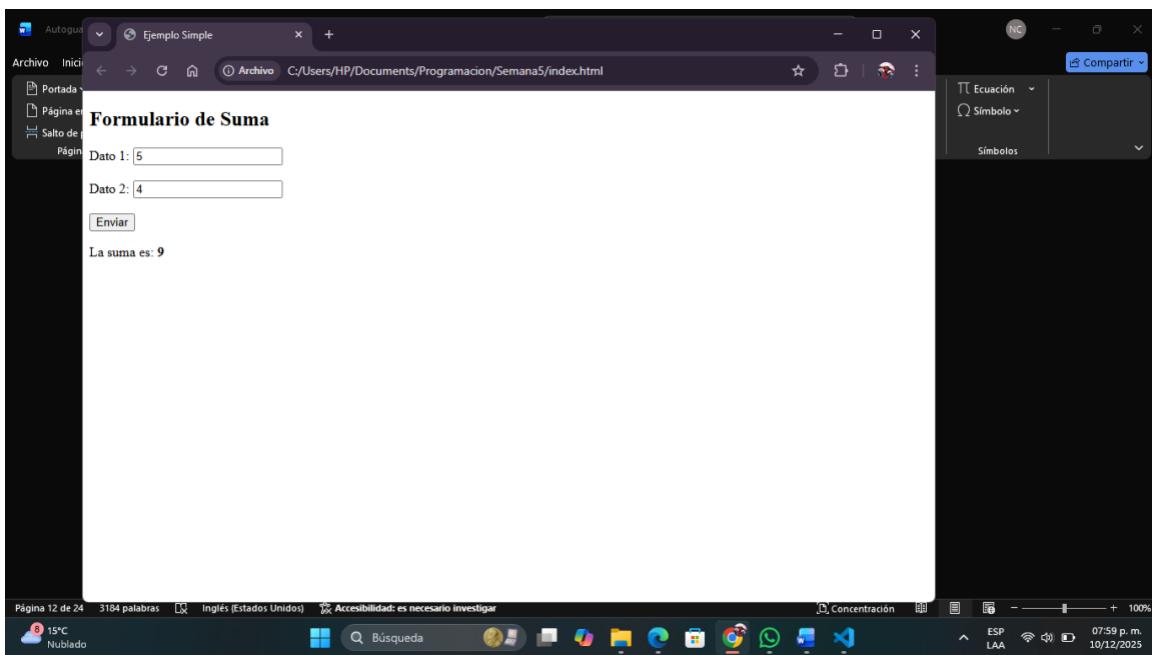
The right pane displays the file `inner.html` with the following content:

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<h1>Hola este es un ejemplo de como funciona el innerHTML</h1>
<h2>Primera prueba llenar los espacios de una lista</h2>
<p>Lorem ipsum dolor sit amet consectetur adipisicing elit.</p>
<ul>
<li id="idea1"> </li>
<li id="idea2"> </li>
<li id="idea3"> </li>
<li id="idea4"> </li>
<li id="idea5"> </li>
</ul>
<button onclick="llenarLista()">Da Clic para llenar<br>
<h3 id="idea6"></h3>
<h1>Ejemplo 2 Solicitar un nombre y saludarlo</h1>
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit.</p>
<input type="text" id="nombre" placeholder="Escribe tu nombre...">
<button onclick="saludar()">Saludar</button>
<h3 id="misaludo"></h3>
<h1>Ejemplo 3 usando document.getElementbyTagName</h1>
<input type="text" name="nombre" placeholder="Escribe tu nombre...">
<p>Este es un párrafo de ejemplo 1.</p>
<p>Este es un párrafo de ejemplo 2.</p>
</body>

```

The status bar at the bottom shows the following information: Página 13 de 24, 3184 palabras, Inglés (Estados Unidos), Accesibilidad: es necesario investigar, 0758 p. m., 10/12/2025.



Semana 6 – Estructuras2025

Propósito del módulo

Este directorio corresponde a la Semana 6 del curso “Estructuras2025”. En esta fase se espera que el estudiante consolide los conocimientos previos, aplique estructuras de datos y lógica avanzada, y posiblemente integre conceptos más sofisticados — preparándose para temas más complejos en las semanas siguientes.

Estructura sugerida del directorio (Semana 6)

Lenguaje(s) y formato utilizados

- Lenguaje de programación: probablemente C , C++ (o el que hayas usado en el curso).
- Podrías emplear: código fuente (` .c` , ` .cpp`), cabeceras (` .h`), módulos auxiliares, utilerías comunes, scripts o archivos de prueba.
- Si el proyecto crece, es recomendable mantener modularidad — separando lógica principal, utilerías, pruebas — para mantener orden.

Objetivos de la Semana 6

Para esta semana se busca:

- Realizar ejercicios de complejidad media-alta que pongan a prueba lo aprendido hasta ahora.
- Manejar correctamente estructuras de datos, funciones, memoria, modularización y, si aplica, entradas/salidas o manejo de datos externos.
- Aplicar buenas prácticas: claridad en el código, documentación, organización, separación de responsabilidades.
- Si corresponde: implementar pruebas o casos de test, para verificar la funcionalidad del código bajo diferentes escenarios.
- Preparar la base para temas más avanzados en semanas futuras.

Contenido esperado / Descripción general

Durante la Semana 6, el estudiante podría realizar actividades como:

- Definir e implementar estructuras o algoritmos más elaborados: listas, nodos, estructuras dinámicas, manejo de memoria, etc.
- Crear utilerías o funciones auxiliares reutilizables (por ejemplo: funciones para manipular estructuras, para entrada/salida, validaciones).
- Realizar operaciones complejas: inserciones, eliminaciones, búsquedas, ordenamientos, manejo de datos dinámicos, liberación de memoria, etc.
- Si hay carpeta de pruebas: hacer casos de prueba y verificar los resultados esperados ante distintos inputs.
- Documentar el código: comentarios, explicación de funciones/estructuras, instrucción de compilación/uso.

Análisis y recomendaciones

-  Modularidad y reutilización: Tener utilerías comunes y separar funciones facilita el mantenimiento y la extensión del proyecto.
-  Pruebas y validaciones: Incluir tests, casos de prueba o validaciones ayuda a garantizar que el programa funciona correctamente incluso si hay cambios futuros.
-  Buena documentación: Comentarios claros, descripciones de funciones/estructuras, instrucciones de compilación/uso — esto facilita que otros (o tú en el futuro) entiendan rápidamente el propósito de cada archivo.
-  Consistencia en nomenclatura y estructura: Usar nombres homogéneos (`ejercicio6_1.c`, `utils6.h`, `test6_1.in`, etc.) ayuda a mantener orden y legibilidad.
-  Preparación para lo que viene: Si mantienes buenas prácticas desde ahora, será mucho más fácil integrar estructuras y algoritmos más complejos más adelante sin desorden.

Conclusión

La carpeta `Semana6` del curso “Estructuras2025”, siguiendo esta plantilla, representa un nivel intermedio-avanzado: con ejercicios más elaborados, modularidad, uso de utilerías comunes, posibles pruebas y buena organización. Al completar esta plantilla con tu contenido real, tendrás una documentación clara, ordenada y profesional, ideal para mantener el repositorio organizado y comprensible.

A screenshot of a Windows desktop environment. On the left, there is a file explorer window showing a folder structure. In the center, a code editor window displays two files: 'index.html' and 'JS code.js'. The 'index.html' file contains HTML code for a web page about data structures. The 'JS code.js' file contains JavaScript code for displaying variable types. At the bottom, a terminal window shows the command prompt and the output of the JavaScript code execution.

```
C:\> Users > HP > Documents > Programacion > Semana6 > index.html ...
C:\> Users > HP > Documents > Programacion > Semana6 > JS code.js > mostrarVariables
1 function mostrarVariables(){
2     let nombre = "Juan Pérez"; // String
3     let edad = 30; // Number
4     let esEstudiante = true; // Boolean
5     let estatura = 1.70; // Double
6
7     document.getElementById("resultado").innerHTML =
8         `

Nombre: ${nombre} (Tipo: ${typeof nombre})


9         

Edad: ${edad} (Tipo: ${typeof edad})

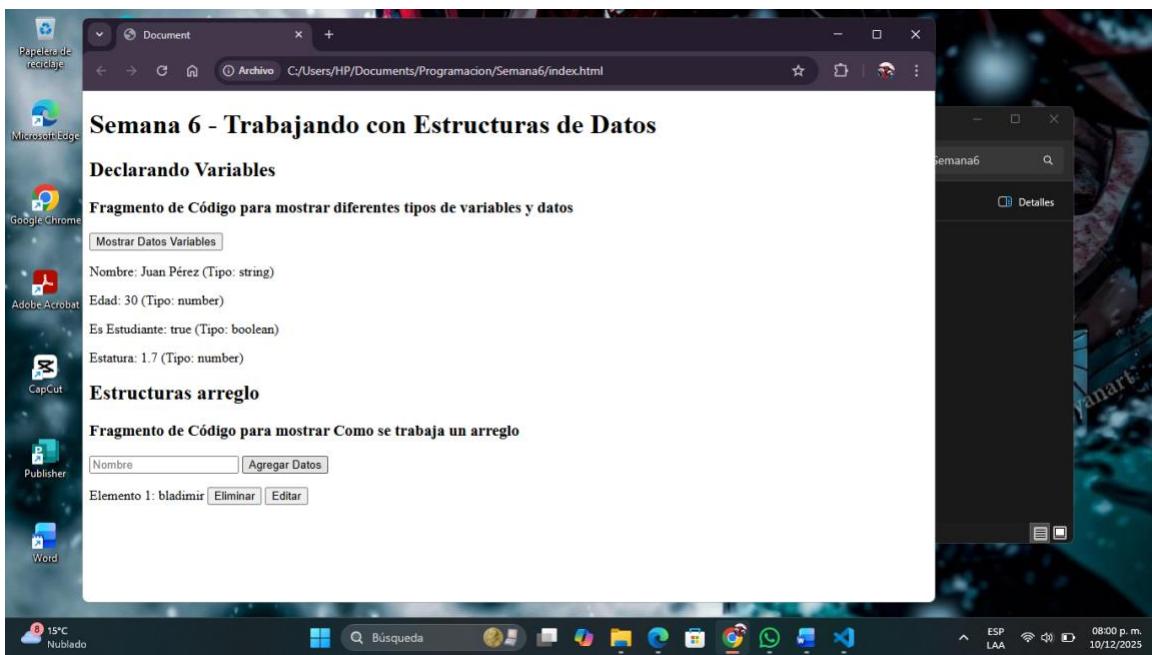

10        

Es Estudiante: ${esEstudiante} (Tipo: ${typeof esEstudiante})


11        

Estatura: ${estatura} (Tipo: ${typeof estatura})


12    }
13
14 let datos = [];
15
16 function datosarreglo(){
17     nombre = document.getElementById("nombre").value;
18     document.getElementById("nombre").value = "";
19
20     datos.push(nombre);
21
22     document.getElementById("elementos").innerHTML = "";
23
24     datos.forEach(function(item, index){
25         document.getElementById("elementos").innerHTML +
26     });
27 }
28
```



Semana 7 – Estructuras2025

Propósito del módulo

La carpeta correspondiente a la Semana 7 del curso “Estructuras2025” está destinada a que el estudiante avance hacia ejercicios de mayor complejidad, consolide estructuras y funciones ya vistas, e integre lógica más robusta. Se espera que al cierre de esta semana, el alumno tenga una base sólida para abordar proyectos o prácticas más amplias.

Estructura sugerida del directorio (Semana 7)

Lenguaje(s) y formato empleados

- Lenguaje de programación: probablemente C , C++ u otro según lo que utilice el curso.
- Uso de: código fuente (.c, ` .cpp`), cabeceras (.h`), módulos auxiliares, utilerías compartidas.
- Si aplica: estructura de pruebas (input/output), modularización clara, separación de responsabilidades.

Objetivos de la Semana 7

Al terminar esta semana, el estudiante debería ser capaz de:

- Abordar ejercicios con estructuras de datos o algoritmos de complejidad intermedia- avanzada.
- Manejar modularidad: usar utilerías auxiliares, dividir lógica entre funciones/módulos, reutilización de código.
- Implementar pruebas o casos de test para validar el correcto funcionamiento del código bajo distintos escenarios.
- Documentar claramente su código: comentarios, instrucciones de compilación/ejecución, uso, especificaciones.
- Mantener organización y buenas prácticas, de cara a proyectos futuros del curso.

Contenido esperado / Descripción general

Durante esta semana 7 podrían realizarse actividades como:

- Diseño e implementación de estructuras de datos más complejas o combinadas — listas, nodos, árboles, punteros, memoria dinámica, según lo aprendido.
- Implementar operaciones sobre esas estructuras: inserción, eliminación, búsqueda, ordenamientos, recorridos, manejo dinámico, liberado de memoria, etc.

- Separar lógica en módulos auxiliares reutilizables: utilerías para manejo de datos, estructuras, operaciones comunes.
- Si es pertinente: crear pruebas automáticas o manuales — con entradas y salidas — para verificar la correcta ejecución del código bajo diferentes escenarios.
- Documentar: describir qué hace cada archivo/módulo, cómo compilar, cómo ejecutar, ejemplos de uso.

Análisis y recomendaciones

-  Modularidad y reutilización : Tener módulos auxiliares mejora mantenibilidad y claridad, sobre todo si el proyecto comienza a crecer.
-  Pruebas / validación : Incluir casos de prueba ayuda a asegurar que el código funciona correctamente incluso al hacer modificaciones o refactorizaciones.
-  Buena documentación : Comentarios, README por semana, instrucciones claras de compilación y uso — esto facilita que tú, u otros, revisen o usen el código en el futuro.
-  Consistencia en nombres y estructura : Mantener una convención clara de nombres de archivos, carpetas, ejercicios, módulos — lo que facilita navegación y mantenimiento.
-  Preparación para lo que sigue : Si ya mantienes organización, modularidad y documentación desde ahora, será mucho más sencillo integrar temas más avanzados o proyectos complejos en semanas futuras.

Conclusión

La carpeta `Semana7` del curso “Estructuras2025”, bajo esta plantilla, representa una etapa intermedia-avanzada: con ejercicios más elaborados, modularidad, utilerías reutilizables, pruebas y buena documentación. Al completar esta plantilla con tus archivos reales, tendrás un README claro, organizado y profesional — útil para ti y para cualquier otra persona que colabore o revise el proyecto.

The screenshot shows a code editor window with two tabs: `index.html` and `code.js`.

index.html:

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, i
6   <title>Document</title>
7 </head>
8 <body>
9   <h1>Semana 7 - Trabajando con Estructuras de Datos</h1>
10  <h2>Declarando Variables</h2>
11  <h3>Fragmento de Código para mostrar diferentes tipo
12  <button onclick="mostrarVariables()">Mostrar Datos V
13  <div id="resultado"></div>
14
15  <h2>Estructuras arreglo</h2>
16  <h3>Fragmento de Código para mostrar Como se trabaja
17  <input type="text" id="nombre" placeholder="Nombre">
18  <button onclick="datosarreglo()">Agregar Datos<butt
19  <div id="elementos">
20  </div>
21
22  <h2>Estructuras de datos multiples</h2>
23  <h3>Fragmento de Código para mostrar Como se trabaja
24  <input type="text" id="nombre1" placeholder="Nombre">
25  <input type="text" id="edad" placeholder="Edad">
26  <input type="tel" id="telefono" placeholder="Telefono">
27  <input type="number" id="es" placeholder="Estatura">
28  <label for="genero">Soltero:</label>
29  <select id="genero">
30    <option value="true">Si</option>
31    <option value="false">No</option>
32  </select>

```

code.js:

```

1 function mostrarVariables(){
2   let nombre = "Juan Pérez"; // String
3   let edad = 30; // Number
4   let esEstudiante = true; // Boolean
5   let estatura = 1.70; // Double
6
7   document.getElementById("resultado").innerHTML =
8     '<p>Nombre: ${nombre} (Tipo: ${typeof nombre})</p>
9     <p>Edad: ${edad} (Tipo: ${typeof edad})</p>
10    <p>Es Estudiante: ${esEstudiante} (Tipo: ${typeof
11      esEstudiante})</p>
12      <p>Estatura: ${estatura} (Tipo: ${typeof estatura})</p>
13  }
14
15  let datos = [];
16
17  function datosarreglo(){
18    nombre = document.getElementById("nombre").value;
19    document.getElementById("nombre").value = "";
20
21    datos.push(nombre);
22
23    document.getElementById("elementos").innerHTML = "";
24
25    datos.forEach(function(item, index){
26      document.getElementById("elementos").innerHTML +
27    });
28
29
30  let miEstructura = {
31    nombre: "",
32    edad: 0,
33  }
34
35  miEstructura.nombre = nombre;
36  miEstructura.edad = edad;
37
38  console.log(miEstructura);
39
40  document.getElementById("resultado").innerHTML =
41    '<p>Nombre: ${miEstructura.nombre} (Tipo: ${typeof
42      miEstructura.nombre})</p>
43      <p>Edad: ${miEstructura.edad} (Tipo: ${typeof
44        miEstructura.edad})</p>
45      <p>Estatura: ${miEstructura.estatura} (Tipo: ${typeof
46        miEstructura.estatura})</p>
47      <p>Soltero: ${miEstructura.soltero} (Tipo: ${typeof
48        miEstructura.soltero})</p>
49      <p>Genero: ${miEstructura.genero} (Tipo: ${typeof
50        miEstructura.genero})</p>
51  '
52}

```

The screenshot shows a browser window displaying the output of the code execution.

Header:

Autogestión Document Document

Content:

Estructuras arreglo

Fragmento de Código para mostrar Como se trabaja un arreglo

Nombre Agregar Datos

Estructuras de datos multiples

Fragmento de Código para mostrar Como se trabaja una Estructura de datos

Bladimir 19 288474747 Soltero: Agregar Datos

Estructuras de datos multiples

Fragmento de Código para mostrar Como se trabaja una Estructura de datos como arreglo

bladimir
19
6373737
1.70
Soltero: Agregar Datos

Página 16 de 26 3184 palabras Inglés (Estados Unidos) Accesibilidad: es necesario investigar

Semana 8 – Estructuras2025

 Módulo: Linked List

Propósito del módulo

Este directorio contiene los ejercicios y código correspondiente a la semana 8 del curso “Estructuras2025”, centrados en el estudio e implementación de listas enlazadas. El objetivo principal es que el estudiante comprenda, implemente y manipule una estructura dinámica de datos tipo lista enlazada, consolidando el uso de punteros, memoria dinámica o referencias según el lenguaje, y operaciones comunes sobre listas.

 Estructura del directorio (linkedlist)

Lenguaje(s) y formato utilizados

- Lenguaje de programación: probablemente C o C++ (dependiendo del curso).
- Uso de estructuras tipo `struct` (en C) o clases/`struct` en C++ para definir nodos de la lista, con campos de dato y puntero al siguiente nodo. Esto es típico en listas enlazadas simples. 
- Separación entre interfaz (cabecera `.h`) y definición/implementación (`.c` / `.`cpp`) — buena práctica para modularidad.
- Si incluyes pruebas: archivos de entrada/salida, o scripts de prueba para verificar el funcionamiento de la lista.

Objetivos del módulo “Linked List”

Al final de este módulo deberías poder:

- Comprender qué es una lista enlazada: estructura de nodos dinámicos conectados mediante punteros / referencias. 
- Implementar las operaciones básicas de una lista enlazada simple: creación/inicialización de la lista, inserción (al inicio / al final o en posiciones intermedias), eliminación, recorrido, búsqueda, impresión, liberación de memoria (si aplica). 
- Manejar correctamente memoria dinámica (o referencias/objetos, según el lenguaje), garantizando que no haya fugas, punteros “colgantes” o errores de acceso tras eliminar nodos.
- Escribir código modular y limpio: separar interfaz y lógica, documentar funciones, comentar código, y estructurar carpetas de forma coherente.

- (Opcional) Crear casos de prueba para validar el comportamiento del código con distintos escenarios: lista vacía, inserción en distintos contextos, eliminación, recorrido, etc.

Descripción del contenido

En este módulo “linkedlist” se trabaja lo siguiente:

Definición de la estructura de nodo: normalmente algo así como:

```c

```
Typedef struct Nodo {
 Int dato; // o dato genérico
 Struct Nodo* siguiente; // apuntador al siguiente nodo (NULL si es el último)
} Nodo;
```

Análisis y recomendaciones

 Uso de listas enlazadas como base de estructuras dinámicas: Las listas enlazadas son fundamentales para muchas estructuras de datos, ya que permiten crecer o reducir dinámicamente sin necesidad de tamaño fijo como en arrays.

 Complejidad y manejo cuidadoso de punteros / memoria dinámica: Al trabajar con punteros (o referencias) es esencial controlar bien la asignación y liberación de memoria, así como evitar punteros colgantes o accesos fuera de límites.

 Documentación y modularidad — clave si el proyecto crece: Separar la interfaz de la implementación, comentar funciones, explicar parámetros y posibles errores hace que el código sea más legible y mantenible.

 Pruebas para robustez: Es recomendable tener varios escenarios de prueba: lista vacía, inserción múltiple, eliminación en cabeza/cola/medio, búsqueda de elementos inexistentes — para asegurar que tu implementación maneja correctamente casos límite y errores.

 Buena práctica de nomenclatura y estructura de carpetas: Usar nombres claros (por ejemplo linkedlist.h, linkedlist.c, main.c, tests/) ayuda a entender rápidamente la organización, facilita colaboración y mantenimiento.

## Conclusión

La carpeta Semana8/linkedlist está destinada a introducir y practicar la implementación de una lista enlazada — una estructura dinámica fundamental en programación. Si completas esta plantilla con tus archivos reales, tendrás una documentación clara y profesional: definición de estructura, funcionalidades, pruebas, y organización modular. Esto facilitará tu aprendizaje y también el mantenimiento o extensión futura del proyecto.

The screenshot shows a Microsoft Edge browser window with two tabs open:

- JS doublyList.js**:

```
1 // Estructura de un singly linked list
2 let doublyLinked = {
3 head: {
4 value: 1,
5 next: {
6 value: 2,
7 next: {
8 value: 3,
9 next: null,
10 // Null para permitir que halla otro
11 },
12 },
13 },
14};
15
16 class Nodes {
17 constructor(value) {
18 // nodo tiene valor
19 this.value = value;
20 this.next = null;
21 this.prev = null;
22 }
23 }
24
25 class myDoublyLinkedList {
26 //value es para iniciar con una cabeza
27 constructor(value) {
28 this.head = {
29 value: value,
30 next: null,
31 prev: null,
32 };
33 }
```
- JS singly.js**:

```
1 let singlyLinked = {
2 head: {
3 value: 1,
4 next: {
5 value: 2,
6 next: {
7 value: 3,
8 next: null,
9 // Null para permitir que halla otro
10 },
11 },
12 },
13 };
14 const recorrer = (lista) => {
15 for (i of lista) {
16 console.log(i);
17 }
18 };
19
20 recorrer(singlyLinked);
21 }
```

The browser interface includes a navigation bar with back, forward, and search buttons, as well as a status bar at the bottom showing the current file path, line number, and zoom level.

The screenshot shows a Microsoft Edge browser window with four tabs open in a 2x2 grid:

- doublyList.js**:

```
1 class Nodes {
2 constructor(value) {
3 // nodo tiene valor
4 this.value = value;
5 this.next = null;
6 }
7 }
8
9 class myDoublyLinkedList {
10 //value es para iniciar con una cabeza
11 constructor(value) {
12 this.head = {
13 value: value,
14 next: null,
15 };
16 // Solo apunta a la cabeza
17 this.tail = this.head;
18 this.length = 1;
19 }
20 }
```
- singly.js**:

```
C:\> Users > HP > Documents > Programacion > Semana8 > linkedlist > singly.js > Nodes
1 class Nodes {
2 constructor(value) {
3 // nodo tiene valor
4 this.value = value;
5 this.next = null;
6 }
7 }
8
9 class myDoublyLinkedList {
10 //value es para iniciar con una cabeza
11 constructor(value) {
12 this.head = {
13 value: value,
14 next: null,
15 };
16 // Solo apunta a la cabeza
17 this.tail = this.head;
18 this.length = 1;
19 }
20 }
```
- code.js**:

```
C:\> Users > HP > Documents > Programacion > Semana8 > linkedlist > code.js > ...
1 let singlyLinked = {
2 head: {
3 value: 1,
4 next: {
5 value: 2,
6 next: {
7 value: 3,
8 next: null,
9 // Null para permitir que hallo otro
10 },
11 },
12 },
13};
14 const recorrer = (lista) => {
15 for (i of lista) {
16 console.log(i);
17 }
18};
19
20 recorrer(singlyLinked);
21
```
- addNode.js**:

```
C:\> Users > HP > Documents > Programacion > Semana8 > linkedlist > addNode.js > ...
1 let singlyLinked = {
2 head: {
3 value: 1,
4 next: {
5 value: 2,
6 next: {
7 value: 3,
8 next: null,
9 // Null para permitir que hallo otro
10 },
11 },
12 },
13};
14 const recorrer = (lista) => {
15 for (i of lista) {
16 console.log(i);
17 }
18};
19
20 recorrer(singlyLinked);
21
```

The browser interface includes a top navigation bar with File, Edit, Selection, View, Go, and a search bar. The bottom status bar shows the page number (Página 18 de 2), zoom level (15%), weather (Nublado), and system information (Ln 1, Col 1, Spaces: 2, UTF-8, LF, JavaScript). The bottom taskbar includes icons for File Explorer, Task View, and several pinned applications.

## Semana 9 – Estructuras2025

### Propósito del módulo

Este directorio contiene los materiales y ejercicios correspondientes a la Semana 9 del curso “Estructuras2025”. En esta semana se espera consolidar lo aprendido hasta ahora y avanzar hacia ejercicios o conceptos más complejos, posiblemente preparando al estudiante para estructuras más avanzadas o combinaciones de estructuras de datos.

### Estructura sugerida del directorio (Semana 9)

### Lenguaje(s) y formato esperados

- Lenguaje de programación principal: el que use el curso (por ejemplo C, C++, u otro).
- Uso de código fuente (`.c`, `.cpp`, etc.), posiblemente archivos de cabecera (`.h`), módulos auxiliares, utilerías comunes, etc.
- Si el proyecto lo requiere: estructura modular, separación de lógica, utilerías comunes y — si aplica — pruebas o casos de test.

### Objetivos de la Semana 9

Durante esta semana se espera que el estudiante:

- Aplique estructuras y conceptos vistos previamente en ejercicios de mayor complejidad.
- Combine conocimientos: por ejemplo, manejo de estructuras de datos, lógica, funciones, modularidad, memoria dinámica (si aplica), entrada/salida, etc.
- Escriba código bien estructurado y documentado: con separación entre interfaz/implementación, funciones modulares, comentarios claros.
- (Opcional) Genere pruebas o conjuntos de test para verificar que el código funcione correctamente en distintos escenarios (casos límite, datos variados, listas vacías, etc.).
- Prepare el terreno para próximas semanas o proyectos finales, consolidando buenas prácticas de desarrollo.

### Contenido esperado / Descripción general

Los ejercicios o actividades que podría incluir esta semana:

- Problemas o tareas que requieren mayor reflexión: manipulación de estructuras de datos complejas, combinaciones de estructuras, transformaciones, algoritmos sobre datos.
- Código modular: utilerías comunes, funciones auxiliares, separación de responsabilidades.
- Manejo adecuado de memoria o recursos (si el lenguaje lo requiere): creación, liberación, validación, manejo de errores.
- (Si aplica) Pruebas automáticas o manuales: entradas/salidas de prueba, validaciones, verificación de resultados esperados.
- Documentación tanto del código como de la estructura del proyecto: comentarios, uso de README, instrucciones para compilar o ejecutar, ejemplos de uso.

## Análisis y recomendaciones

-  Organización modular y clara : separar lógica, utilerías y pruebas ayuda muchísimo cuando el proyecto crece.
-  Uso de pruebas / validaciones : incluir tests o casos de prueba ayuda a asegurar que el código funcione correctamente en distintos escenarios y facilita mantenimiento.
-  Documentación útil : un README por semana, con descripción clara de archivos, ejercicios y cómo compilar/ejecutar, mejora la legibilidad del repositorio. De hecho, mantener documentación con formato Markdown es una buena práctica reconocida. [¶1¶](#)
-  Consistencia en nombres y estructura : usar convenciones claras (por ejemplo `ejercicio9\_1.c`, `utils9.h`, `test9\_1.in`) facilita navegación, colaboración y mantenimiento.
-  Preparación para lo que viene : si mantienes buenas prácticas en esta etapa — modularidad, documentación, tests — tendrás una base sólida para proyectos más avanzados o el desarrollo de estructuras/algoritmos más complejos.

## Conclusión

La carpeta `Semana9` del curso “Estructuras2025” representa una etapa de consolidación y transición hacia ejercicios más complejos, combinando lo aprendido de semanas anteriores. Si completas esta plantilla con tus archivos reales, tendrás una documentación clara, organizada y profesional — útil tanto para ti como para quien revise o colabore en el proyecto.

The screenshot shows a code editor with two tabs open: `graph.js` and `hash_table.js`. The left pane displays `graph.js`, which contains code for a `Graph` class. The right pane displays `hash_table.js`, which contains code for a `HashTable` class.

```
graph.js
1 /**
2 *
3 * 2 - 0
4 * * / \
5 * * 1 - 3
6 */
7 class Graph{
8 constructor(){
9 this.nodos = 0;
10 this.adjacentList = {};
11 }
12
13 addVertices(node){
14 this.adjacentList[node] = [];
15 this.nodos++;
16 }
17
18 addEdge(node1, node2){
19 this.adjacentList[node1].push(node2);
20 this.adjacentList[node2].push(node1);
21 }
22}
23
24 const myGraph = new Graph();
25 myGraph.addVertices("1");
26 console.log("Agrego nodo ", myGraph);
27 myGraph.addVertices("3");
28 console.log("Agrego nodo ", myGraph);
29 myGraph.addVertices("4");
30 console.log("Agrego nodo ", myGraph);
31 myGraph.addVertices("5");
32 console.log("Agrego nodo ", myGraph);
```

```
hash_table.js
1 class HashTable{
2 constructor(size){
3 this.data = new Array(size);
4 }
5
6 hashMethod(key){
7 let hash = 0;
8 for(let i = 0; i<key.length; i++){
9 hash = (hash + key.charCodeAt(i)*i) % this.data.length;
10 }
11
12 return hash;
13 }
14
15 set(key,value){
16 const address = this.hashMethod(key);
17 if(!this.data[address]){
18 this.data[address] = [];
19 }
20
21 this.data[address].push([key,value]);
22 }
23
24 get (key){
25 const address = this.hashMethod(key);
26
27 const currentBucket = this.data[address];
28
29 if(currentBucket){
30 for(let i = 0; i< currentBucket.length; i++){
31 if(currentBucket[i][0] == key){
32 return currentBucket[i][1];
33 }
34 }
35 }
36 }
37}
```

The screenshot shows a dual-pane code editor interface. The left pane displays the file `singly_linkedList.js` with the following code:

```
1 class Node{
2 constructor(value){
3 this.value = value;
4 this.next = null;
5 }
6 }
7
8 class MySinglyLinkedList {
9 constructor(value){
10 this.head = {
11 value: value,
12 next: null
13 }
14
15 this.tail = this.head;
16
17 this.length = 1;
18 }
19
20
21 append(value){
22 const newNode = new Node(value);
23
24 this.tail.next = newNode;
25 this.tail = newNode;
26
27 this.length++;
28
29 return this
30
31 }
32}
```

The right pane displays the file `hash_table.js` with the following code:

```
1 /*+
2 * 18
3 * 4 20
4 * 2 8 17 170
5 */
6
7
8 class Node{
9 constructor(value){
10 this.left = null;
11 this.right = null;
12 this.value = value
13 }
14
15
16 class BinarySearchTree{
17 constructor(){
18 this.root = null
19 }
20
21 insert(value){
22 const newNode = new Node(value);
23
24 if(this.root === null){
25 this.root = newNode;
26 }else{
27 let currentNode = this.root;
28
29 while(true){
30 if(value < currentNode.value){
31 if(!currentNode.left){
32 currentNode.left = newNode;
33 }else{
34 currentNode = currentNode.left;
35 }
36 }else{
37 if(!currentNode.right){
38 currentNode.right = newNode;
39 }else{
40 currentNode = currentNode.right;
41 }
42 }
43 }
44 }
45 }
46
47 search(value){
48 let currentNode = this.root;
49
50 while(currentNode !== null){
51 if(value === currentNode.value){
52 return true;
53 }else if(value < currentNode.value){
54 currentNode = currentNode.left;
55 }else{
56 currentNode = currentNode.right;
57 }
58 }
59
60 return false;
61 }
62
63 delete(value){
64 let currentNode = this.root;
65 let parent = null;
66
67 while(currentNode !== null){
68 if(value === currentNode.value){
69 if(currentNode.left === null && currentNode.right === null){
70 if(parent === null){
71 this.root = null;
72 }else{
73 if(parent.left === currentNode){
74 parent.left = null;
75 }else{
76 parent.right = null;
77 }
78 }
79 }else if(currentNode.left === null){
80 if(parent === null){
81 this.root = currentNode.right;
82 }else{
83 if(parent.left === currentNode){
84 parent.left = currentNode.right;
85 }else{
86 parent.right = currentNode.right;
87 }
88 }
89 }else if(currentNode.right === null){
90 if(parent === null){
91 this.root = currentNode.left;
92 }else{
93 if(parent.left === currentNode){
94 parent.left = currentNode.left;
95 }else{
96 parent.right = currentNode.left;
97 }
98 }
99 }else{
100 let successor = currentNode.right;
101 let successorParent = currentNode;
102
103 while(successor.left !== null){
104 successorParent = successor;
105 successor = successor.left;
106 }
107
108 if(successorParent === currentNode){
109 successorParent.right = null;
110 }else{
111 successorParent.left = null;
112 }
113
114 currentNode.value = successor.value;
115 }
116 }
117 if(currentNode === value){
118 break;
119 }
120 if(currentNode.value < value){
121 parent = currentNode;
122 currentNode = currentNode.right;
123 }else{
124 parent = currentNode;
125 currentNode = currentNode.left;
126 }
127 }
128 }
129
130 printInorder(){
131 let currentNode = this.root;
132
133 while(currentNode !== null){
134 if(currentNode.left !== null){
135 currentNode = currentNode.left;
136 }else{
137 console.log(currentNode.value);
138 if(currentNode.right !== null){
139 currentNode = currentNode.right;
140 }
141 }
142 }
143 }
144
145 printPreorder(){
146 let currentNode = this.root;
147
148 while(currentNode !== null){
149 console.log(currentNode.value);
150 if(currentNode.left !== null){
151 currentNode = currentNode.left;
152 }else{
153 if(currentNode.right !== null){
154 currentNode = currentNode.right;
155 }
156 }
157 }
158 }
159
160 printPostorder(){
161 let currentNode = this.root;
162
163 while(currentNode !== null){
164 if(currentNode.right !== null){
165 currentNode = currentNode.right;
166 }else{
167 console.log(currentNode.value);
168 if(currentNode.left !== null){
169 currentNode = currentNode.left;
170 }
171 }
172 }
173 }
174
175 printLevelOrder(){
176 let queue = [this.root];
177
178 while(queue.length > 0){
179 let node = queue.shift();
180
181 if(node !== null){
182 console.log(node.value);
183 if(node.left !== null){
184 queue.push(node.left);
185 }
186 if(node.right !== null){
187 queue.push(node.right);
188 }
189 }
190 }
191 }
192
193 printLevelOrder2(){
194 let queue = [this.root];
195
196 while(queue.length > 0){
197 let levelSize = queue.length;
198 let currentLevel = [];
199
200 for(let i = 0; i < levelSize; i++){
201 let node = queue.shift();
202
203 if(node !== null){
204 currentLevel.push(node.value);
205 if(node.left !== null){
206 queue.push(node.left);
207 }
208 if(node.right !== null){
209 queue.push(node.right);
210 }
211 }
212 }
213
214 if(currentLevel.length > 0){
215 console.log(currentLevel);
216 }
217 }
218 }
219
220 printLevelOrder3(){
221 let queue = [this.root];
222
223 while(queue.length > 0){
224 let levelSize = queue.length;
225 let currentLevel = [];
226
227 for(let i = 0; i < levelSize; i++){
228 let node = queue.shift();
229
230 if(node !== null){
231 currentLevel.push(node.value);
232 if(node.left !== null){
233 queue.push(node.left);
234 }
235 if(node.right !== null){
236 queue.push(node.right);
237 }
238 }
239 }
240
241 if(currentLevel.length > 0){
242 console.log(currentLevel);
243 }
244 }
245 }
246
247 printLevelOrder4(){
248 let queue = [this.root];
249
250 while(queue.length > 0){
251 let levelSize = queue.length;
252 let currentLevel = [];
253
254 for(let i = 0; i < levelSize; i++){
255 let node = queue.shift();
256
257 if(node !== null){
258 currentLevel.push(node.value);
259 if(node.left !== null){
260 queue.push(node.left);
261 }
262 if(node.right !== null){
263 queue.push(node.right);
264 }
265 }
266 }
267
268 if(currentLevel.length > 0){
269 console.log(currentLevel);
270 }
271 }
272 }
273
274 printLevelOrder5(){
275 let queue = [this.root];
276
277 while(queue.length > 0){
278 let levelSize = queue.length;
279 let currentLevel = [];
280
281 for(let i = 0; i < levelSize; i++){
282 let node = queue.shift();
283
284 if(node !== null){
285 currentLevel.push(node.value);
286 if(node.left !== null){
287 queue.push(node.left);
288 }
289 if(node.right !== null){
290 queue.push(node.right);
291 }
292 }
293 }
294
295 if(currentLevel.length > 0){
296 console.log(currentLevel);
297 }
298 }
299 }
300
301 printLevelOrder6(){
302 let queue = [this.root];
303
304 while(queue.length > 0){
305 let levelSize = queue.length;
306 let currentLevel = [];
307
308 for(let i = 0; i < levelSize; i++){
309 let node = queue.shift();
310
311 if(node !== null){
312 currentLevel.push(node.value);
313 if(node.left !== null){
314 queue.push(node.left);
315 }
316 if(node.right !== null){
317 queue.push(node.right);
318 }
319 }
320 }
321
322 if(currentLevel.length > 0){
323 console.log(currentLevel);
324 }
325 }
326 }
327
328 printLevelOrder7(){
329 let queue = [this.root];
330
331 while(queue.length > 0){
332 let levelSize = queue.length;
333 let currentLevel = [];
334
335 for(let i = 0; i < levelSize; i++){
336 let node = queue.shift();
337
338 if(node !== null){
339 currentLevel.push(node.value);
340 if(node.left !== null){
341 queue.push(node.left);
342 }
343 if(node.right !== null){
344 queue.push(node.right);
345 }
346 }
347 }
348
349 if(currentLevel.length > 0){
350 console.log(currentLevel);
351 }
352 }
353 }
354
355 printLevelOrder8(){
356 let queue = [this.root];
357
358 while(queue.length > 0){
359 let levelSize = queue.length;
360 let currentLevel = [];
361
362 for(let i = 0; i < levelSize; i++){
363 let node = queue.shift();
364
365 if(node !== null){
366 currentLevel.push(node.value);
367 if(node.left !== null){
368 queue.push(node.left);
369 }
370 if(node.right !== null){
371 queue.push(node.right);
372 }
373 }
374 }
375
376 if(currentLevel.length > 0){
377 console.log(currentLevel);
378 }
379 }
380 }
381
382 printLevelOrder9(){
383 let queue = [this.root];
384
385 while(queue.length > 0){
386 let levelSize = queue.length;
387 let currentLevel = [];
388
389 for(let i = 0; i < levelSize; i++){
390 let node = queue.shift();
391
392 if(node !== null){
393 currentLevel.push(node.value);
394 if(node.left !== null){
395 queue.push(node.left);
396 }
397 if(node.right !== null){
398 queue.push(node.right);
399 }
400 }
401 }
402
403 if(currentLevel.length > 0){
404 console.log(currentLevel);
405 }
406 }
407 }
408
409 printLevelOrder10(){
410 let queue = [this.root];
411
412 while(queue.length > 0){
413 let levelSize = queue.length;
414 let currentLevel = [];
415
416 for(let i = 0; i < levelSize; i++){
417 let node = queue.shift();
418
419 if(node !== null){
420 currentLevel.push(node.value);
421 if(node.left !== null){
422 queue.push(node.left);
423 }
424 if(node.right !== null){
425 queue.push(node.right);
426 }
427 }
428 }
429
430 if(currentLevel.length > 0){
431 console.log(currentLevel);
432 }
433 }
434 }
435
436 printLevelOrder11(){
437 let queue = [this.root];
438
439 while(queue.length > 0){
440 let levelSize = queue.length;
441 let currentLevel = [];
442
443 for(let i = 0; i < levelSize; i++){
444 let node = queue.shift();
445
446 if(node !== null){
447 currentLevel.push(node.value);
448 if(node.left !== null){
449 queue.push(node.left);
450 }
451 if(node.right !== null){
452 queue.push(node.right);
453 }
454 }
455 }
456
457 if(currentLevel.length > 0){
458 console.log(currentLevel);
459 }
460 }
461 }
462
463 printLevelOrder12(){
464 let queue = [this.root];
465
466 while(queue.length > 0){
467 let levelSize = queue.length;
468 let currentLevel = [];
469
470 for(let i = 0; i < levelSize; i++){
471 let node = queue.shift();
472
473 if(node !== null){
474 currentLevel.push(node.value);
475 if(node.left !== null){
476 queue.push(node.left);
477 }
478 if(node.right !== null){
479 queue.push(node.right);
480 }
481 }
482 }
483
484 if(currentLevel.length > 0){
485 console.log(currentLevel);
486 }
487 }
488 }
489
490 printLevelOrder13(){
491 let queue = [this.root];
492
493 while(queue.length > 0){
494 let levelSize = queue.length;
495 let currentLevel = [];
496
497 for(let i = 0; i < levelSize; i++){
498 let node = queue.shift();
499
500 if(node !== null){
501 currentLevel.push(node.value);
502 if(node.left !== null){
503 queue.push(node.left);
504 }
505 if(node.right !== null){
506 queue.push(node.right);
507 }
508 }
509 }
510
511 if(currentLevel.length > 0){
512 console.log(currentLevel);
513 }
514 }
515 }
516
517 printLevelOrder14(){
518 let queue = [this.root];
519
520 while(queue.length > 0){
521 let levelSize = queue.length;
522 let currentLevel = [];
523
524 for(let i = 0; i < levelSize; i++){
525 let node = queue.shift();
526
527 if(node !== null){
528 currentLevel.push(node.value);
529 if(node.left !== null){
530 queue.push(node.left);
531 }
532 if(node.right !== null){
533 queue.push(node.right);
534 }
535 }
536 }
537
538 if(currentLevel.length > 0){
539 console.log(currentLevel);
540 }
541 }
542 }
543
544 printLevelOrder15(){
545 let queue = [this.root];
546
547 while(queue.length > 0){
548 let levelSize = queue.length;
549 let currentLevel = [];
550
551 for(let i = 0; i < levelSize; i++){
552 let node = queue.shift();
553
554 if(node !== null){
555 currentLevel.push(node.value);
556 if(node.left !== null){
557 queue.push(node.left);
558 }
559 if(node.right !== null){
560 queue.push(node.right);
561 }
562 }
563 }
564
565 if(currentLevel.length > 0){
566 console.log(currentLevel);
567 }
568 }
569 }
570
571 printLevelOrder16(){
572 let queue = [this.root];
573
574 while(queue.length > 0){
575 let levelSize = queue.length;
576 let currentLevel = [];
577
578 for(let i = 0; i < levelSize; i++){
579 let node = queue.shift();
580
581 if(node !== null){
582 currentLevel.push(node.value);
583 if(node.left !== null){
584 queue.push(node.left);
585 }
586 if(node.right !== null){
587 queue.push(node.right);
588 }
589 }
590 }
591
592 if(currentLevel.length > 0){
593 console.log(currentLevel);
594 }
595 }
596 }
597
598 printLevelOrder17(){
599 let queue = [this.root];
600
601 while(queue.length > 0){
602 let levelSize = queue.length;
603 let currentLevel = [];
604
605 for(let i = 0; i < levelSize; i++){
606 let node = queue.shift();
607
608 if(node !== null){
609 currentLevel.push(node.value);
610 if(node.left !== null){
611 queue.push(node.left);
612 }
613 if(node.right !== null){
614 queue.push(node.right);
615 }
616 }
617 }
618
619 if(currentLevel.length > 0){
620 console.log(currentLevel);
621 }
622 }
623 }
624
625 printLevelOrder18(){
626 let queue = [this.root];
627
628 while(queue.length > 0){
629 let levelSize = queue.length;
630 let currentLevel = [];
631
632 for(let i = 0; i < levelSize; i++){
633 let node = queue.shift();
634
635 if(node !== null){
636 currentLevel.push(node.value);
637 if(node.left !== null){
638 queue.push(node.left);
639 }
640 if(node.right !== null){
641 queue.push(node.right);
642 }
643 }
644 }
645
646 if(currentLevel.length > 0){
647 console.log(currentLevel);
648 }
649 }
650 }
651
652 printLevelOrder19(){
653 let queue = [this.root];
654
655 while(queue.length > 0){
656 let levelSize = queue.length;
657 let currentLevel = [];
658
659 for(let i = 0; i < levelSize; i++){
660 let node = queue.shift();
661
662 if(node !== null){
663 currentLevel.push(node.value);
664 if(node.left !== null){
665 queue.push(node.left);
666 }
667 if(node.right !== null){
668 queue.push(node.right);
669 }
670 }
671 }
672
673 if(currentLevel.length > 0){
674 console.log(currentLevel);
675 }
676 }
677 }
678
679 printLevelOrder20(){
680 let queue = [this.root];
681
682 while(queue.length > 0){
683 let levelSize = queue.length;
684 let currentLevel = [];
685
686 for(let i = 0; i < levelSize; i++){
687 let node = queue.shift();
688
689 if(node !== null){
690 currentLevel.push(node.value);
691 if(node.left !== null){
692 queue.push(node.left);
693 }
694 if(node.right !== null){
695 queue.push(node.right);
696 }
697 }
698 }
699
700 if(currentLevel.length > 0){
701 console.log(currentLevel);
702 }
703 }
704 }
705
706 printLevelOrder21(){
707 let queue = [this.root];
708
709 while(queue.length > 0){
710 let levelSize = queue.length;
711 let currentLevel = [];
712
713 for(let i = 0; i < levelSize; i++){
714 let node = queue.shift();
715
716 if(node !== null){
717 currentLevel.push(node.value);
718 if(node.left !== null){
719 queue.push(node.left);
720 }
721 if(node.right !== null){
722 queue.push(node.right);
723 }
724 }
725 }
726
727 if(currentLevel.length > 0){
728 console.log(currentLevel);
729 }
730 }
731 }
732
733 printLevelOrder22(){
734 let queue = [this.root];
735
736 while(queue.length > 0){
737 let levelSize = queue.length;
738 let currentLevel = [];
739
740 for(let i = 0; i < levelSize; i++){
741 let node = queue.shift();
742
743 if(node !== null){
744 currentLevel.push(node.value);
745 if(node.left !== null){
746 queue.push(node.left);
747 }
748 if(node.right !== null){
749 queue.push(node.right);
750 }
751 }
752 }
753
754 if(currentLevel.length > 0){
755 console.log(currentLevel);
756 }
757 }
758 }
759
760 printLevelOrder23(){
761 let queue = [this.root];
762
763 while(queue.length > 0){
764 let levelSize = queue.length;
765 let currentLevel = [];
766
767 for(let i = 0; i < levelSize; i++){
768 let node = queue.shift();
769
770 if(node !== null){
771 currentLevel.push(node.value);
772 if(node.left !== null){
773 queue.push(node.left);
774 }
775 if(node.right !== null){
776 queue.push(node.right);
777 }
778 }
779 }
780
781 if(currentLevel.length > 0){
782 console.log(currentLevel);
783 }
784 }
785 }
786
787 printLevelOrder24(){
788 let queue = [this.root];
789
790 while(queue.length > 0){
791 let levelSize = queue.length;
792 let currentLevel = [];
793
794 for(let i = 0; i < levelSize; i++){
795 let node = queue.shift();
796
797 if(node !== null){
798 currentLevel.push(node.value);
799 if(node.left !== null){
800 queue.push(node.left);
801 }
802 if(node.right !== null){
803 queue.push(node.right);
804 }
805 }
806 }
807
808 if(currentLevel.length > 0){
809 console.log(currentLevel);
810 }
811 }
812 }
813
814 printLevelOrder25(){
815 let queue = [this.root];
816
817 while(queue.length > 0){
818 let levelSize = queue.length;
819 let currentLevel = [];
820
821 for(let i = 0; i < levelSize; i++){
822 let node = queue.shift();
823
824 if(node !== null){
825 currentLevel.push(node.value);
826 if(node.left !== null){
827 queue.push(node.left);
828 }
829 if(node.right !== null){
830 queue.push(node.right);
831 }
832 }
833 }
834
835 if(currentLevel.length > 0){
836 console.log(currentLevel);
837 }
838 }
839 }
840
841 printLevelOrder26(){
842 let queue = [this.root];
843
844 while(queue.length > 0){
845 let levelSize = queue.length;
846 let currentLevel = [];
847
848 for(let i = 0; i < levelSize; i++){
849 let node = queue.shift();
850
851 if(node !== null){
852 currentLevel.push(node.value);
853 if(node.left !== null){
854 queue.push(node.left);
855 }
856 if(node.right !== null){
857 queue.push(node.right);
858 }
859 }
860 }
861
862 if(currentLevel.length > 0){
863 console.log(currentLevel);
864 }
865 }
866 }
867
868 printLevelOrder27(){
869 let queue = [this.root];
870
871 while(queue.length > 0){
872 let levelSize = queue.length;
873 let currentLevel = [];
874
875 for(let i = 0; i < levelSize; i++){
876 let node = queue.shift();
877
878 if(node !== null){
879 currentLevel.push(node.value);
880 if(node.left !== null){
881 queue.push(node.left);
882 }
883 if(node.right !== null){
884 queue.push(node.right);
885 }
886 }
887 }
888
889 if(currentLevel.length > 0){
890 console.log(currentLevel);
891 }
892 }
893 }
894
895 printLevelOrder28(){
896 let queue = [this.root];
897
898 while(queue.length > 0){
899 let levelSize = queue.length;
900 let currentLevel = [];
901
902 for(let i = 0; i < levelSize; i++){
903 let node = queue.shift();
904
905 if(node !== null){
906 currentLevel.push(node.value);
907 if(node.left !== null){
908 queue.push(node.left);
909 }
910 if(node.right !== null){
911 queue.push(node.right);
912 }
913 }
914 }
915
916 if(currentLevel.length > 0){
917 console.log(currentLevel);
918 }
919 }
920 }
921
922 printLevelOrder29(){
923 let queue = [this.root];
924
925 while(queue.length > 0){
926 let levelSize = queue.length;
927 let currentLevel = [];
928
929 for(let i = 0; i < levelSize; i++){
930 let node = queue.shift();
931
932 if(node !== null){
933 currentLevel.push(node.value);
934 if(node.left !== null){
935 queue.push(node.left);
936 }
937 if(node.right !== null){
938 queue.push(node.right);
939 }
940 }
941 }
942
943 if(currentLevel.length > 0){
944 console.log(currentLevel);
945 }
946 }
947 }
948
949 printLevelOrder30(){
950 let queue = [this.root];
951
952 while(queue.length > 0){
953 let levelSize = queue.length;
954 let currentLevel = [];
955
956 for(let i = 0; i < levelSize; i++){
957 let node = queue.shift();
958
959 if(node !== null){
960 currentLevel.push(node.value);
961 if(node.left !== null){
962 queue.push(node.left);
963 }
964 if(node.right !== null){
965 queue.push(node.right);
966 }
967 }
968 }
969
970 if(currentLevel.length > 0){
971 console.log(currentLevel);
972 }
973 }
974 }
975
976 printLevelOrder31(){
977 let queue = [this.root];
978
979 while(queue.length > 0){
980 let levelSize = queue.length;
981 let currentLevel = [];
982
983 for(let i = 0; i < levelSize; i++){
984 let node = queue.shift();
985
986 if(node !== null){
987 currentLevel.push(node.value);
988 if(node.left !== null){
989 queue.push(node.left);
990 }
991 if(node.right !== null){
992 queue.push(node.right);
993 }
994 }
995 }
996
997 if(currentLevel.length > 0){
998 console.log(currentLevel);
999 }
1000 }
1001 }
1002}
```

## Semana 10 – Estructuras2025

### Propósito del módulo

Este directorio contiene los materiales y ejercicios correspondientes a la Semana 10 del curso “Estructuras2025”. En esta fase se espera que el estudiante consolide conceptos, aplique estructuras y lógica más avanzadas, y despliegue trabajos más completos o integrados — posiblemente combinando varios elementos vistos en semanas anteriores.

#### Estructura sugerida del directorio (Semana 10)

### Lenguaje(s) y formato utilizados

- Lenguaje de programación: probablemente C , C++ u otro según lo que hayas usado en el curso.
- Uso de: código fuente (`.c`, `.cpp`, etc.), posiblemente cabeceras (`.h`), módulos auxiliares, utilerías comunes, y si aplica, pruebas o scripts.
- Organización modular: separación entre lógica principal, utilerías comunes, módulos auxiliares, carpetas de tests — lo que favorece legibilidad y mantenimiento.

### Objetivos de la Semana 10

Para esta semana se busca que:

- Apliques de forma integrada los conocimientos adquiridos hasta ahora: estructuras de datos, algoritmos, modularidad, gestión de memoria (si aplica), lectura/escritura, etc.
- Desarrollos ejercicios o proyectos de complejidad mayor, capaces de combinar varias operaciones: por ejemplo, manipulación de datos, estructuras dinámicas, validación, casos especiales, etc.
- Organices tu código de forma clara y modular: con utilerías reutilizables, separación lógica, documentación interna (comentarios, cabeceras, etc.).
- Si es pertinente: implementes pruebas — con entradas y salidas — para verificar que tu código se comporta correctamente en distintos escenarios.
- Preparación para proyectos futuros o tareas finales del curso — con bases sólidas, buen estilo de código y estructura clara.

## Contenido esperado / Descripción general

Durante la Semana 10, podrías trabajar en:

- Problemas más complejos: por ejemplo, combinaciones de estructuras de datos, manipulación de listas, nodos, memoria dinámica, datos dinámicos, algoritmos de ordenamiento/búsqueda, etc.
- Programas completos o pequeños sistemas: uso de varias funciones, estructuras y módulos — no solo ejercicios aislados.
- Utilerías comunes: funciones reutilizables para operaciones frecuentes (inserción/eliminación/búsqueda en estructuras, manejo de errores, validaciones, etc.).
- Manejo de memoria o recursos (si usas un lenguaje como C/C++): asignación, liberación, cuidado con punteros, evitar fugas.
- (Si decides incluir) casos de prueba: entradas variadas, escenarios límite, validación de resultados esperados.
- Documentación interna: comentarios, explicación de cada módulo/función, instrucciones para compilar/ejecutar (si aplican), ejemplos o notas de uso.

## Análisis y recomendaciones

-  Modularidad y reutilización son clave: A medida que el proyecto se vuelve más complejo, separar utilerías, lógica y pruebas ayuda a mantener el código limpio y manejable.
-  Pruebas / validaciones fortalecen la robustez del programa: probar distintos casos (casos límite, entradas incorrectas, estructuras vacías, etc.) te ayuda a detectar errores y asegurar funcionamiento correcto.
-  Documentación clara mejora la mantenibilidad: Comentarios, descripciones en las cabeceras, README por módulo— todo ayuda si alguien más revisa tu código, o si lo retomas después de tiempo. Además, un buen README actúa como “puerta de entrada” para comprender el propósito y uso del módulo. [¶1¶](#)
-  Convenciones consistentes ayudan mucho: Nombres claros para archivos, estructura de carpetas uniforme, patrones de nombres consistentes (ej. `ejercicio10\_1.c`, `utils10.h`, `tests/`, etc.) facilitan navegación y mantenimiento.
-  Preparación para ampliaciones o integración: Este tipo de organización te permitirá expandir el proyecto — agregar nuevas funcionalidades, mejorar código, integrar con otros módulos — sin desordenar.

## Conclusión

La carpeta `Semana10` del curso “Estructuras2025”, usando esta plantilla, representa una etapa avanzada/intermedia-avanzada: con ejercicios complejos, posible integración de varios conceptos, modularidad, pruebas, documentación y una estructura clara. Si completas esta plantilla con tus archivos reales, tendrás una documentación profesional y útil — tanto para ti como para quien revise o colabore en el proyecto.

The screenshot shows a code editor with two tabs open. The left tab is 'index.html' containing the HTML code for a Rick and Morty demo page. The right tab is 'JS linkedList.js' containing the JavaScript code for a linked list implementation.

```

index.html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <link rel="stylesheet" href="style.css">
7 <title>Rick and Morty - Demo</title>
8 </head>
9 <body>
10 <header class="site-header">
11 <div class="container">
12 <div class="brand">
13 <div class="logo-placeholder">RM</div>
14 </div>
15 <div class="site-title">Rick & Morty</div>
16 <div class="muted" style="font-size: small; margin-top: 10px;">
17 Explora la API
18 <p>Esta demo carga los personajes de la API pública de Rick y Morty y muestra tarjetas responsivas con imagen, estado y otros metadatos</p>
19 </div>
20 <div class="nav-actions">
21 Rick and Morty
22 </div>
23 </header>
24
25 <main>
26 <section class="hero">
27 <div class="container hero-grid">
28 <div class="hero-card">
29 <h1 style="margin: 0 8px">Explora la API</h1>
30 <p>Esta demo carga los personajes de la API pública de Rick y Morty y muestra tarjetas responsivas con imagen, estado y otros metadatos</p>
31 </div>
32 </section>

```

```

JS linkedList.js
1 // Linked Lists
2 // Methods:
3 // push()
4 // pop()
5 // get(index)
6 // delete(index) -> Not implemented yet
7 // isEmpty()
8
9 class Node {
10 constructor(value) {
11 this.value = value;
12 this.next = null;
13 }
14 }
15
16 class LinkedList {
17 constructor() {
18 this.head = null;
19 this.tail = null;
20 this.length = 0;
21 }
22
23 isEmpty() {
24 return this.length === 0;
25 }
26
27 push(value) {
28 const newNode = new Node(value);
29
30 if (this.isEmpty()) {
31 this.head = newNode;
32 this.tail = newNode;
33 } else {
34 this.tail.next = newNode;
35 this.tail = newNode;
36 }
37 this.length++;
38 }
39
40 pop() {
41 if (this.isEmpty()) {
42 return null;
43 }
44 let currentHead = this.head;
45 let previousHead = null;
46
47 while (currentHead !== null) {
48 previousHead = currentHead;
49 currentHead = currentHead.next;
50 }
51
52 this.head = previousHead;
53 this.tail = previousHead;
54
55 this.length--;
56
57 return currentHead;
58 }
59
60 get(index) {
61 if (index < 0 || index >= this.length) {
62 return null;
63 }
64
65 let currentHead = this.head;
66 let currentIndex = 0;
67
68 while (currentIndex !== index) {
69 currentHead = currentHead.next;
70 currentIndex++;
71 }
72
73 return currentHead;
74 }
75
76 delete(index) {
77 if (index < 0 || index >= this.length) {
78 return null;
79 }
80
81 let currentHead = this.head;
82 let previousHead = null;
83 let currentIndex = 0;
84
85 while (currentIndex !== index) {
86 previousHead = currentHead;
87 currentHead = currentHead.next;
88 currentIndex++;
89 }
90
91 if (previousHead === null) {
92 this.head = currentHead;
93 } else {
94 previousHead.next = currentHead;
95 }
96
97 this.length--;
98
99 return currentHead;
100 }
101
102 isEmpty() {
103 return this.length === 0;
104 }
105
106 length() {
107 return this.length;
108 }
109
110 print() {
111 let currentHead = this.head;
112 let result = '';
113
114 while (currentHead !== null) {
115 result += currentHead.value + ' ';
116 currentHead = currentHead.next;
117 }
118
119 console.log(result);
120 }
121
122 printReverse() {
123 let currentHead = this.head;
124 let result = '';
125
126 while (currentHead !== null) {
127 result += currentHead.value + ' ';
128 currentHead = currentHead.next;
129 }
130
131 console.log(result);
132 }
133
134 reverse() {
135 let currentHead = this.head;
136 let previousHead = null;
137
138 while (currentHead !== null) {
139 let nextHead = currentHead.next;
140 currentHead.next = previousHead;
141 previousHead = currentHead;
142 currentHead = nextHead;
143 }
144
145 this.head = previousHead;
146 }
147
148 size() {
149 return this.length;
150 }
151
152 toArray() {
153 let currentHead = this.head;
154 let result = [];
155
156 while (currentHead !== null) {
157 result.push(currentHead.value);
158 currentHead = currentHead.next;
159 }
160
161 return result;
162 }
163
164 fromArray(array) {
165 let currentHead = this.head;
166
167 while (currentHead !== null) {
168 currentHead = currentHead.next;
169 }
170
171 for (let i = 0; i < array.length; i++) {
172 this.push(array[i]);
173 }
174
175 this.length = array.length;
176 }
177
178 clear() {
179 this.head = null;
180 this.tail = null;
181 this.length = 0;
182 }
183
184 toJSON() {
185 let currentHead = this.head;
186 let result = [];
187
188 while (currentHead !== null) {
189 result.push(currentHead.value);
190 currentHead = currentHead.next;
191 }
192
193 return result;
194 }
195
196 fromJSON(json) {
197 let currentHead = this.head;
198
199 while (currentHead !== null) {
200 currentHead = currentHead.next;
201 }
202
203 for (let i = 0; i < json.length; i++) {
204 this.push(json[i]);
205 }
206
207 this.length = json.length;
208 }
209
210 copy() {
211 let currentHead = this.head;
212 let result = new LinkedList();
213
214 while (currentHead !== null) {
215 result.push(currentHead.value);
216 currentHead = currentHead.next;
217 }
218
219 return result;
220 }
221
222 clone() {
223 let currentHead = this.head;
224 let result = new LinkedList();
225
226 while (currentHead !== null) {
227 result.push(currentHead.value);
228 currentHead = currentHead.next;
229 }
230
231 return result;
232 }
233
234 equals(otherList) {
235 if (this.length !== otherList.length) {
236 return false;
237 }
238
239 let currentHead = this.head;
240 let otherHead = otherList.head;
241
242 while (currentHead !== null) {
243 if (currentHead.value !== otherHead.value) {
244 return false;
245 }
246
247 currentHead = currentHead.next;
248 otherHead = otherHead.next;
249 }
250
251 return true;
252 }
253
254 hashCode() {
255 let currentHead = this.head;
256 let hash = 0;
257
258 while (currentHead !== null) {
259 hash = hash * 31 + currentHead.value;
260 currentHead = currentHead.next;
261 }
262
263 return hash;
264 }
265
266 toString() {
267 let currentHead = this.head;
268 let result = '';
269
270 while (currentHead !== null) {
271 result += currentHead.value + ' ';
272 currentHead = currentHead.next;
273 }
274
275 return result;
276 }
277
278 toObject() {
279 let currentHead = this.head;
280 let result = {};
281
282 while (currentHead !== null) {
283 result[currentHead.value] = currentHead.next;
284 currentHead = currentHead.next;
285 }
286
287 return result;
288 }
289
290 toList() {
291 let currentHead = this.head;
292 let result = [];
293
294 while (currentHead !== null) {
295 result.push(currentHead.value);
296 currentHead = currentHead.next;
297 }
298
299 return result;
300 }
301
302 toSet() {
303 let currentHead = this.head;
304 let result = new Set();
305
306 while (currentHead !== null) {
307 result.add(currentHead.value);
308 currentHead = currentHead.next;
309 }
310
311 return result;
312 }
313
314 toMap() {
315 let currentHead = this.head;
316 let result = new Map();
317
318 while (currentHead !== null) {
319 result.set(currentHead.value, currentHead.next);
320 currentHead = currentHead.next;
321 }
322
323 return result;
324 }
325
326 toPriorityQueue() {
327 let currentHead = this.head;
328 let result = new PriorityQueue();
329
330 while (currentHead !== null) {
331 result.insert(currentHead.value);
332 currentHead = currentHead.next;
333 }
334
335 return result;
336 }
337
338 toGraph() {
339 let currentHead = this.head;
340 let result = new Graph();
341
342 while (currentHead !== null) {
343 result.addNode(currentHead.value);
344 currentHead = currentHead.next;
345 }
346
347 let currentHead2 = this.head;
348 let currentHead3 = this.head;
349
350 while (currentHead2 !== null) {
351 let currentHead4 = this.head;
352
353 while (currentHead4 !== null) {
354 if (currentHead2.value === currentHead4.value) {
355 result.addEdge(currentHead2.value, currentHead4.value);
356 }
357
358 currentHead4 = currentHead4.next;
359 }
360
361 currentHead2 = currentHead2.next;
362 }
363
364 return result;
365 }
366
367 toBFS() {
368 let currentHead = this.head;
369 let result = new BFS();
370
371 while (currentHead !== null) {
372 result.addNode(currentHead.value);
373 currentHead = currentHead.next;
374 }
375
376 let currentHead2 = this.head;
377
378 while (currentHead2 !== null) {
379 let currentHead3 = this.head;
380
381 while (currentHead3 !== null) {
382 if (currentHead2.value === currentHead3.value) {
383 result.addEdge(currentHead2.value, currentHead3.value);
384 }
385
386 currentHead3 = currentHead3.next;
387 }
388
389 currentHead2 = currentHead2.next;
390 }
391
392 return result;
393 }
394
395 toDFS() {
396 let currentHead = this.head;
397 let result = new DFS();
398
399 while (currentHead !== null) {
400 result.addNode(currentHead.value);
401 currentHead = currentHead.next;
402 }
403
404 let currentHead2 = this.head;
405
406 while (currentHead2 !== null) {
407 let currentHead3 = this.head;
408
409 while (currentHead3 !== null) {
410 if (currentHead2.value === currentHead3.value) {
411 result.addEdge(currentHead2.value, currentHead3.value);
412 }
413
414 currentHead3 = currentHead3.next;
415 }
416
417 currentHead2 = currentHead2.next;
418 }
419
420 return result;
421 }
422
423 toDFTS() {
424 let currentHead = this.head;
425 let result = new DFTS();
426
427 while (currentHead !== null) {
428 result.addNode(currentHead.value);
429 currentHead = currentHead.next;
430 }
431
432 let currentHead2 = this.head;
433
434 while (currentHead2 !== null) {
435 let currentHead3 = this.head;
436
437 while (currentHead3 !== null) {
438 if (currentHead2.value === currentHead3.value) {
439 result.addEdge(currentHead2.value, currentHead3.value);
440 }
441
442 currentHead3 = currentHead3.next;
443 }
444
445 currentHead2 = currentHead2.next;
446 }
447
448 return result;
449 }
450
451 toBFS() {
452 let currentHead = this.head;
453 let result = new BFS();
454
455 while (currentHead !== null) {
456 result.addNode(currentHead.value);
457 currentHead = currentHead.next;
458 }
459
460 let currentHead2 = this.head;
461
462 while (currentHead2 !== null) {
463 let currentHead3 = this.head;
464
465 while (currentHead3 !== null) {
466 if (currentHead2.value === currentHead3.value) {
467 result.addEdge(currentHead2.value, currentHead3.value);
468 }
469
470 currentHead3 = currentHead3.next;
471 }
472
473 currentHead2 = currentHead2.next;
474 }
475
476 return result;
477 }
478
479 toDFS() {
480 let currentHead = this.head;
481 let result = new DFS();
482
483 while (currentHead !== null) {
484 result.addNode(currentHead.value);
485 currentHead = currentHead.next;
486 }
487
488 let currentHead2 = this.head;
489
490 while (currentHead2 !== null) {
491 let currentHead3 = this.head;
492
493 while (currentHead3 !== null) {
494 if (currentHead2.value === currentHead3.value) {
495 result.addEdge(currentHead2.value, currentHead3.value);
496 }
497
498 currentHead3 = currentHead3.next;
499 }
500
501 currentHead2 = currentHead2.next;
502 }
503
504 return result;
505 }
506
507 toDFTS() {
508 let currentHead = this.head;
509 let result = new DFTS();
510
511 while (currentHead !== null) {
512 result.addNode(currentHead.value);
513 currentHead = currentHead.next;
514 }
515
516 let currentHead2 = this.head;
517
518 while (currentHead2 !== null) {
519 let currentHead3 = this.head;
520
521 while (currentHead3 !== null) {
522 if (currentHead2.value === currentHead3.value) {
523 result.addEdge(currentHead2.value, currentHead3.value);
524 }
525
526 currentHead3 = currentHead3.next;
527 }
528
529 currentHead2 = currentHead2.next;
530 }
531
532 return result;
533 }
534
535 toBFS() {
536 let currentHead = this.head;
537 let result = new BFS();
538
539 while (currentHead !== null) {
540 result.addNode(currentHead.value);
541 currentHead = currentHead.next;
542 }
543
544 let currentHead2 = this.head;
545
546 while (currentHead2 !== null) {
547 let currentHead3 = this.head;
548
549 while (currentHead3 !== null) {
550 if (currentHead2.value === currentHead3.value) {
551 result.addEdge(currentHead2.value, currentHead3.value);
552 }
553
554 currentHead3 = currentHead3.next;
555 }
556
557 currentHead2 = currentHead2.next;
558 }
559
560 return result;
561 }
562
563 toDFS() {
564 let currentHead = this.head;
565 let result = new DFS();
566
567 while (currentHead !== null) {
568 result.addNode(currentHead.value);
569 currentHead = currentHead.next;
570 }
571
572 let currentHead2 = this.head;
573
574 while (currentHead2 !== null) {
575 let currentHead3 = this.head;
576
577 while (currentHead3 !== null) {
578 if (currentHead2.value === currentHead3.value) {
579 result.addEdge(currentHead2.value, currentHead3.value);
580 }
581
582 currentHead3 = currentHead3.next;
583 }
584
585 currentHead2 = currentHead2.next;
586 }
587
588 return result;
589 }
590
591 toDFTS() {
592 let currentHead = this.head;
593 let result = new DFTS();
594
595 while (currentHead !== null) {
596 result.addNode(currentHead.value);
597 currentHead = currentHead.next;
598 }
599
600 let currentHead2 = this.head;
601
602 while (currentHead2 !== null) {
603 let currentHead3 = this.head;
604
605 while (currentHead3 !== null) {
606 if (currentHead2.value === currentHead3.value) {
607 result.addEdge(currentHead2.value, currentHead3.value);
608 }
609
610 currentHead3 = currentHead3.next;
611 }
612
613 currentHead2 = currentHead2.next;
614 }
615
616 return result;
617 }
618
619 toBFS() {
620 let currentHead = this.head;
621 let result = new BFS();
622
623 while (currentHead !== null) {
624 result.addNode(currentHead.value);
625 currentHead = currentHead.next;
626 }
627
628 let currentHead2 = this.head;
629
630 while (currentHead2 !== null) {
631 let currentHead3 = this.head;
632
633 while (currentHead3 !== null) {
634 if (currentHead2.value === currentHead3.value) {
635 result.addEdge(currentHead2.value, currentHead3.value);
636 }
637
638 currentHead3 = currentHead3.next;
639 }
640
641 currentHead2 = currentHead2.next;
642 }
643
644 return result;
645 }
646
647 toDFS() {
648 let currentHead = this.head;
649 let result = new DFS();
650
651 while (currentHead !== null) {
652 result.addNode(currentHead.value);
653 currentHead = currentHead.next;
654 }
655
656 let currentHead2 = this.head;
657
658 while (currentHead2 !== null) {
659 let currentHead3 = this.head;
660
661 while (currentHead3 !== null) {
662 if (currentHead2.value === currentHead3.value) {
663 result.addEdge(currentHead2.value, currentHead3.value);
664 }
665
666 currentHead3 = currentHead3.next;
667 }
668
669 currentHead2 = currentHead2.next;
670 }
671
672 return result;
673 }
674
675 toDFTS() {
676 let currentHead = this.head;
677 let result = new DFTS();
678
679 while (currentHead !== null) {
680 result.addNode(currentHead.value);
681 currentHead = currentHead.next;
682 }
683
684 let currentHead2 = this.head;
685
686 while (currentHead2 !== null) {
687 let currentHead3 = this.head;
688
689 while (currentHead3 !== null) {
690 if (currentHead2.value === currentHead3.value) {
691 result.addEdge(currentHead2.value, currentHead3.value);
692 }
693
694 currentHead3 = currentHead3.next;
695 }
696
697 currentHead2 = currentHead2.next;
698 }
699
700 return result;
701 }
702
703 toBFS() {
704 let currentHead = this.head;
705 let result = new BFS();
706
707 while (currentHead !== null) {
708 result.addNode(currentHead.value);
709 currentHead = currentHead.next;
710 }
711
712 let currentHead2 = this.head;
713
714 while (currentHead2 !== null) {
715 let currentHead3 = this.head;
716
717 while (currentHead3 !== null) {
718 if (currentHead2.value === currentHead3.value) {
719 result.addEdge(currentHead2.value, currentHead3.value);
720 }
721
722 currentHead3 = currentHead3.next;
723 }
724
725 currentHead2 = currentHead2.next;
726 }
727
728 return result;
729 }
730
731 toDFS() {
732 let currentHead = this.head;
733 let result = new DFS();
734
735 while (currentHead !== null) {
736 result.addNode(currentHead.value);
737 currentHead = currentHead.next;
738 }
739
740 let currentHead2 = this.head;
741
742 while (currentHead2 !== null) {
743 let currentHead3 = this.head;
744
745 while (currentHead3 !== null) {
746 if (currentHead2.value === currentHead3.value) {
747 result.addEdge(currentHead2.value, currentHead3.value);
748 }
749
750 currentHead3 = currentHead3.next;
751 }
752
753 currentHead2 = currentHead2.next;
754 }
755
756 return result;
757 }
758
759 toDFTS() {
760 let currentHead = this.head;
761 let result = new DFTS();
762
763 while (currentHead !== null) {
764 result.addNode(currentHead.value);
765 currentHead = currentHead.next;
766 }
767
768 let currentHead2 = this.head;
769
770 while (currentHead2 !== null) {
771 let currentHead3 = this.head;
772
773 while (currentHead3 !== null) {
774 if (currentHead2.value === currentHead3.value) {
775 result.addEdge(currentHead2.value, currentHead3.value);
776 }
777
778 currentHead3 = currentHead3.next;
779 }
780
781 currentHead2 = currentHead2.next;
782 }
783
784 return result;
785 }
786
787 toBFS() {
788 let currentHead = this.head;
789 let result = new BFS();
790
791 while (currentHead !== null) {
792 result.addNode(currentHead.value);
793 currentHead = currentHead.next;
794 }
795
796 let currentHead2 = this.head;
797
798 while (currentHead2 !== null) {
799 let currentHead3 = this.head;
800
801 while (currentHead3 !== null) {
802 if (currentHead2.value === currentHead3.value) {
803 result.addEdge(currentHead2.value, currentHead3.value);
804 }
805
806 currentHead3 = currentHead3.next;
807 }
808
809 currentHead2 = currentHead2.next;
810 }
811
812 return result;
813 }
814
815 toDFS() {
816 let currentHead = this.head;
817 let result = new DFS();
818
819 while (currentHead !== null) {
820 result.addNode(currentHead.value);
821 currentHead = currentHead.next;
822 }
823
824 let currentHead2 = this.head;
825
826 while (currentHead2 !== null) {
827 let currentHead3 = this.head;
828
829 while (currentHead3 !== null) {
830 if (currentHead2.value === currentHead3.value) {
831 result.addEdge(currentHead2.value, currentHead3.value);
832 }
833
834 currentHead3 = currentHead3.next;
835 }
836
837 currentHead2 = currentHead2.next;
838 }
839
840 return result;
841 }
842
843 toDFTS() {
844 let currentHead = this.head;
845 let result = new DFTS();
846
847 while (currentHead !== null) {
848 result.addNode(currentHead.value);
849 currentHead = currentHead.next;
850 }
851
852 let currentHead2 = this.head;
853
854 while (currentHead2 !== null) {
855 let currentHead3 = this.head;
856
857 while (currentHead3 !== null) {
858 if (currentHead2.value === currentHead3.value) {
859 result.addEdge(currentHead2.value, currentHead3.value);
860 }
861
862 currentHead3 = currentHead3.next;
863 }
864
865 currentHead2 = currentHead2.next;
866 }
867
868 return result;
869 }
870
871 toBFS() {
872 let currentHead = this.head;
873 let result = new BFS();
874
875 while (currentHead !== null) {
876 result.addNode(currentHead.value);
877 currentHead = currentHead.next;
878 }
879
880 let currentHead2 = this.head;
881
882 while (currentHead2 !== null) {
883 let currentHead3 = this.head;
884
885 while (currentHead3 !== null) {
886 if (currentHead2.value === currentHead3.value) {
887 result.addEdge(currentHead2.value, currentHead3.value);
888 }
889
890 currentHead3 = currentHead3.next;
891 }
892
893 currentHead2 = currentHead2.next;
894 }
895
896 return result;
897 }
898
899 toDFS() {
900 let currentHead = this.head;
901 let result = new DFS();
902
903 while (currentHead !== null) {
904 result.addNode(currentHead.value);
905 currentHead = currentHead.next;
906 }
907
908 let currentHead2 = this.head;
909
910 while (currentHead2 !== null) {
911 let currentHead3 = this.head;
912
913 while (currentHead3 !== null) {
914 if (currentHead2.value === currentHead3.value) {
915 result.addEdge(currentHead2.value, currentHead3.value);
916 }
917
918 currentHead3 = currentHead3.next;
919 }
920
921 currentHead2 = currentHead2.next;
922 }
923
924 return result;
925 }
926
927 toDFTS() {
928 let currentHead = this.head;
929 let result = new DFTS();
930
931 while (currentHead !== null) {
932 result.addNode(currentHead.value);
933 currentHead = currentHead.next;
934 }
935
936 let currentHead2 = this.head;
937
938 while (currentHead2 !== null) {
939 let currentHead3 = this.head;
940
941 while (currentHead3 !== null) {
942 if (currentHead2.value === currentHead3.value) {
943 result.addEdge(currentHead2.value, currentHead3.value);
944 }
945
946 currentHead3 = currentHead3.next;
947 }
948
949 currentHead2 = currentHead2.next;
950 }
951
952 return result;
953 }
954
955 toBFS() {
956 let currentHead = this.head;
957 let result = new BFS();
958
959 while (currentHead !== null) {
960 result.addNode(currentHead.value);
961 currentHead = currentHead.next;
962 }
963
964 let currentHead2 = this.head;
965
966 while (currentHead2 !== null) {
967 let currentHead3 = this.head;
968
969 while (currentHead3 !== null) {
970 if (currentHead2.value === currentHead3.value) {
971 result.addEdge(currentHead2.value, currentHead3.value);
972 }
973
974 currentHead3 = currentHead3.next;
975 }
976
977 currentHead2 = currentHead2.next;
978 }
979
980 return result;
981 }
982
983 toDFS() {
984 let currentHead = this.head;
985 let result = new DFS();
986
987 while (currentHead !== null) {
988 result.addNode(currentHead.value);
989 currentHead = currentHead.next;
990 }
991
992 let currentHead2 = this.head;
993
994 while (currentHead2 !== null) {
995 let currentHead3 = this.head;
996
997 while (currentHead3 !== null) {
998 if (currentHead2.value === currentHead3.value) {
999 result.addEdge(currentHead2.value, currentHead3.value);
1000 }
1001
1002 currentHead3 = currentHead3.next;
1003 }
1004
1005 currentHead2 = currentHead2.next;
1006 }
1007
1008 return result;
1009 }
1010
1011 toDFTS() {
1012 let currentHead = this.head;
1013 let result = new DFTS();
1014
1015 while (currentHead !== null) {
1016 result.addNode(currentHead.value);
1017 currentHead = currentHead.next;
1018 }
1019
1020 let currentHead2 = this.head;
1021
1022 while (currentHead2 !== null) {
1023 let currentHead3 = this.head;
1024
1025 while (currentHead3 !== null) {
1026 if (currentHead2.value === currentHead3.value) {
1027 result.addEdge(currentHead2.value, currentHead3.value);
1028 }
1029
1030 currentHead3 = currentHead3.next;
1031 }
1032
1033 currentHead2 = currentHead2.next;
1034 }
1035
1036 return result;
1037 }
1038
1039 toBFS() {
1040 let currentHead = this.head;
1041 let result = new BFS();
1042
1043 while (currentHead !== null) {
1044 result.addNode(currentHead.value);
1045 currentHead = currentHead.next;
1046 }
1047
1048 let currentHead2 = this.head;
1049
1050 while (currentHead2 !== null) {
1051 let currentHead3 = this.head;
1052
1053 while (currentHead3 !== null) {
1054 if (currentHead2.value === currentHead3.value) {
1055 result.addEdge(currentHead2.value, currentHead3.value);
1056 }
1057
1058 currentHead3 = currentHead3.next;
1059 }
1060
1061 currentHead2 = currentHead2.next;
1062 }
1063
1064 return result;
1065 }
1066
1067 toDFS() {
1068 let currentHead = this.head;
1069 let result = new DFS();
1070
1071 while (currentHead !== null) {
1072 result.addNode(currentHead.value);
1073 currentHead = currentHead.next;
1074 }
1075
1076 let currentHead2 = this.head;
1077
1078 while (currentHead2 !== null) {
1079 let currentHead3 = this.head;
1080
1081 while (currentHead3 !== null) {
1082 if (currentHead2.value === currentHead3.value) {
1083 result.addEdge(currentHead2.value, currentHead3.value);
1084 }
1085
1086 currentHead3 = currentHead3.next;
1087 }
1088
1089 currentHead2 = currentHead2.next;
1090 }
1091
1092 return result;
1093 }
1094
1095 toDFTS() {
1096 let currentHead = this.head;
1097 let result = new DFTS();
1098
1099 while (currentHead !== null) {
1100 result.addNode(currentHead.value);
1101 currentHead = currentHead.next;
1102 }
1103
1104 let currentHead2 = this.head;
1105
1106 while (currentHead2 !== null) {
1107 let currentHead3 = this.head;
1108
1109 while (currentHead3 !== null) {
1110 if (currentHead2.value === currentHead3.value) {
1111 result.addEdge(currentHead2.value, currentHead3.value);
1112 }
1113
1114 currentHead3 = currentHead3.next;
1115 }
1116
1117 currentHead2 = currentHead2.next;
1118 }
1119
1120 return result;
1121 }
1122
1123 toBFS() {
1124 let currentHead = this.head;
1125 let result = new BFS();
1126
1127 while (currentHead !== null) {
1128 result.addNode(currentHead.value);
1129 currentHead = currentHead.next;
1130 }
1131
1132 let currentHead2 = this.head;
1133
1134 while (currentHead2 !== null) {
1135 let currentHead3 = this.head;
1136
1137 while (currentHead3 !== null) {
1138 if (currentHead2.value === currentHead3.value) {
1139 result.addEdge(currentHead2.value, currentHead3.value);
1140 }
1141
1142 currentHead3 = currentHead3.next;
1143 }
1144
1145 currentHead2 = currentHead2.next;
1146 }
1147
1148 return result;
1149 }
1150
1151 toDFS() {
1152 let currentHead = this.head;
1153 let result = new DFS();
1154
1155 while (currentHead !== null) {
1156 result.addNode(currentHead.value);
1157 currentHead = currentHead.next;
1158 }
1159
1160 let currentHead2 = this.head;
1161
1162 while (currentHead2 !== null) {
1163 let currentHead3 = this.head;
1164
1165 while (currentHead3 !== null) {
1166 if (currentHead2.value === currentHead3.value) {
1167 result.addEdge(currentHead2.value, currentHead3.value);
1168 }
1169
1170 currentHead3 = currentHead3.next;
1171 }
1172
1173 currentHead2 = currentHead2.next;
1174 }
1175
1176 return result;
1177 }
1178
1179 toDFTS() {
1180 let currentHead = this.head;
1181 let result = new DFTS();
1182
1183 while (currentHead !== null) {
1184 result.addNode(currentHead.value);
1185 currentHead = currentHead.next;
1186 }
1187
1188 let currentHead2 = this.head;
1189
1190 while (currentHead2 !== null) {
1191 let currentHead3 = this.head;
1192
1193 while (currentHead3 !== null) {
1194 if (currentHead2.value === currentHead3.value) {
1195 result.addEdge(currentHead2.value, currentHead3.value);
1196 }
1197
1198 currentHead3 = currentHead3.next;
1199 }
1200
1201 currentHead2 = currentHead2.next;
1202 }
1203
1204 return result;
1205 }
1206
1207 toBFS() {
1208 let currentHead = this.head;
1209 let result = new BFS();
1210
1211 while (currentHead !== null) {
1212 result.addNode(currentHead.value);
1213 currentHead = currentHead.next;
1214 }
1215
1216 let currentHead2 = this.head;
1217
1218 while (currentHead2 !== null) {
1219 let currentHead3 = this.head;
1220
1221 while (currentHead3 !== null) {
1222 if (currentHead2.value === currentHead3.value) {
1223 result.addEdge(currentHead2.value, currentHead3.value);
1224 }
1225
1226 currentHead3 = currentHead3.next;
1227 }
1228
1229 currentHead2 = currentHead2.next;
1230 }
1231
1232 return result;
1233 }
1234
1235 toDFS() {
1236 let currentHead = this.head;
1237 let result = new DFS();
1238
1239 while (currentHead !== null) {
1240 result.addNode(currentHead.value);
1241 currentHead = currentHead.next;
1242 }
1243
1244 let currentHead2 = this.head;
1245
1246 while (currentHead2 !== null) {
1247 let currentHead3 = this.head;
1248
1249 while (currentHead3 !== null) {
1250 if (currentHead2.value === currentHead3.value) {
1251 result.addEdge(currentHead2.value, currentHead3.value);
1252 }
1253
1254 currentHead3 = currentHead3.next;
1255 }
1256
1257 currentHead2 = currentHead2.next;
1258 }
1259
1260 return result;
1261 }
1262
1263 toDFTS() {
1264 let currentHead = this.head;
1265 let result = new DFTS();
1266
1267 while (currentHead !== null) {
1268 result.addNode(currentHead.value);
1269 currentHead = currentHead.next;
1270 }
1271
1272 let currentHead2 = this.head;
1273
1274 while (currentHead2 !== null) {
1275 let currentHead3 = this.head;
1276
1277 while (currentHead3 !== null) {
1278 if (currentHead2.value === currentHead3.value) {
1279 result.addEdge(currentHead2.value, currentHead3.value);
1280 }
1281
1282 currentHead3 = currentHead3.next;
1283 }

```

## Semana 11 – Estructuras2025

### Propósito del módulo

Este directorio corresponde a la Semana 11 del curso “Estructuras2025”. En esta etapa, se espera que el estudiante consolide lo aprendido hasta ahora, aplique estructuras de datos y algoritmos de forma integrada, y posiblemente comience a trabajar en ejercicios más complejos o proyectos que requieran combinar múltiples conceptos vistos en semanas anteriores.

### Estructura sugerida del directorio (Semana 11)

### Lenguaje(s) y formato esperados

- Lenguaje de programación utilizado en el curso (por ejemplo C, C++, u otro).
- Uso de código fuente (`.c`, `.cpp`, etc.), posiblemente con archivos de cabecera (`.h`) si se emplea modularidad.
- Si se ha requerido modularización o reutilización de código: presencia de carpetas auxiliares (`modulo\_aux/`), utilerías compartidas, etc.
- Opcionalmente, estructura para pruebas: entradas/salidas de test, validaciones, para asegurar que los ejercicios funcionan correctamente ante distintos escenarios.

### Objetivos de la Semana 11

Durante esta semana, los objetivos principales podrían incluir:

- Integrar múltiples conceptos previamente aprendidos — estructuras de datos, manejo de memoria, algoritmos, modularidad — en ejercicios de mayor complejidad.
- Diseñar código organizado y reutilizable: separar lógica en módulos, usar funciones auxiliares, y mantener buen estilo de programación.
- Implementar funciones o programas completos que combinén varias operaciones (inserción, eliminación, búsqueda, ordenamiento, manipulación dinámica de datos, etc.).
- Validar el funcionamiento mediante pruebas o casos de test, para manejar diferentes escenarios, entradas válidas o inválidas, listas vacías, estructuras dinámicas, etc.
- Documentar bien el código y la estructura del módulo: describir qué hace cada archivo, cómo compilar/ejecutar, cómo usar las funciones, posibles limitaciones o advertencias.

## Contenido esperado / Descripción general

Algunos ejemplos de lo que la semana podría incluir:

- Ejercicios o proyectos que utilicen estructuras de datos dinámicas (listas enlazadas, árboles, colas, colas circulares, etc.) o combinaciones de estructuras.
- Implementación de algoritmos para manipular, buscar, ordenar, recorrer o transformar datos — aprovechando lo aprendido en semanas anteriores.
- Creación de utilerías o funciones comunes (por ejemplo: funciones de manejo de nodos, de lectura/escritura, validación de datos, manejo de errores, liberación de memoria, etc.).
- Si decides usar pruebas: conjuntos de entrada/salida para verificar comportamiento correcto en distintos escenarios (casos estándar, límite, errores, lista vacía, etc.).
- Documentación clara — en comentarios y en README — para que cualquier persona (o tú en un futuro) entienda el propósito, uso y funcionamiento del código.

## Análisis y recomendaciones

-  Modularidad y reutilización: A medida que aumente la complejidad de los ejercicios, mantener funciones auxiliares y estructura clara ayuda en mantenimiento, extensión y claridad del proyecto.
-  Uso de pruebas / validaciones: Incluir tests — con entradas variadas y casos límite — ayuda a detectar errores y asegurar que el programa responde bien ante distintos escenarios.
-  Documentación clara: Comentarios en código, README por módulo, explicación de funciones y uso — todo esto mejora la legibilidad y facilita compartir el proyecto o retomarlo después. Las buenas README suelen contener secciones “qué hace”, “cómo usarlo”, “estructura de archivos”, “requisitos”. 
-  Aplicación práctica de conceptos de estructuras de datos: Si la semana incluye implementación de estructuras como listas enlazadas, árboles, colas, etc., es útil recordar sus propiedades: por ejemplo, las listas enlazadas ofrecen flexibilidad dinámica, pero su acceso aleatorio es costoso. 
-  Consistencia en convenios de nombres y estilo: Usar convenios claros para nombres de archivos, funciones, módulos (por ejemplo: `ejercicio11\_1.c`, `utils11.h`, `tests/`) facilita navegación, colaboración y mantenimiento.

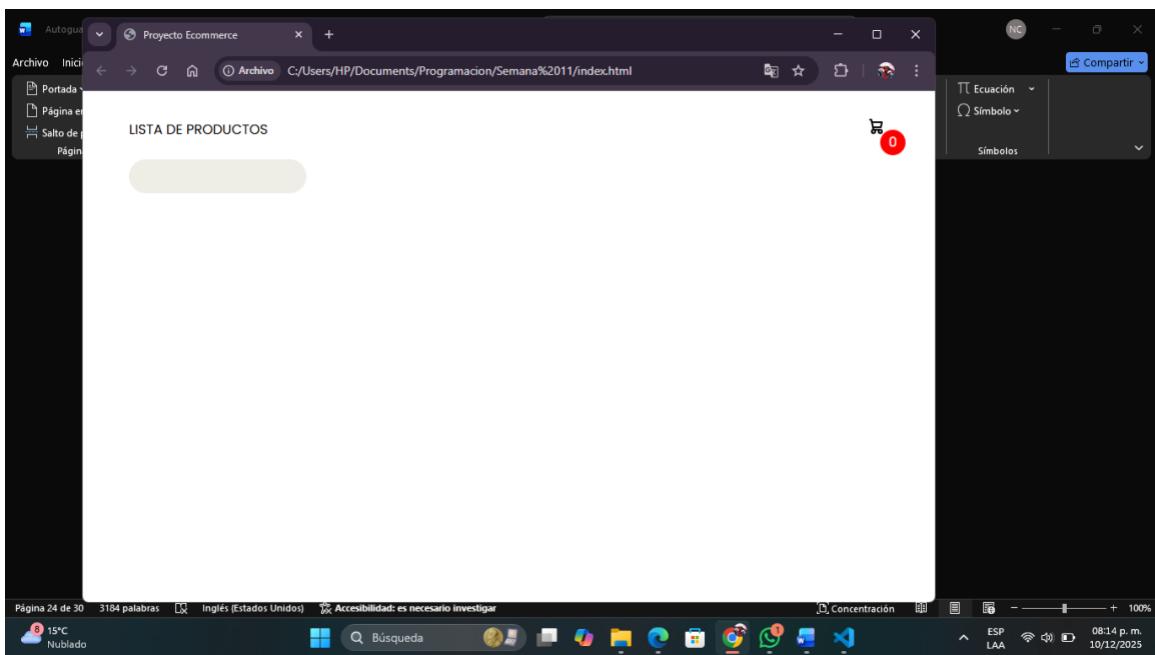
## Conclusión

La carpeta 'Semana11` del curso "Estructuras2025", usando esta plantilla, puede representar una etapa clave de consolidación o transición hacia ejercicios/puntos finales del curso: con contenidos integrados, modularidad, pruebas y documentación. Al completar esta plantilla con tus archivos reales, tendrás una buena documentación, orden, y claridad tanto para ti como para quien revise o colabore en el proyecto.

The screenshot shows a code editor with two tabs: 'index.html' and 'app.js'. The 'index.html' tab displays the structure of an HTML page with sections for a title, a header containing a logo icon, a product list, and a shopping cart tab. The 'app.js' tab shows a script that handles the cart icon's click event to toggle its visibility and adds products to the cart list.

```
index.html
C:\Users\HP\Documents\Programacion\Semana 11> index.html ...
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <!-- Importar el estilo css-->
7 <link rel="stylesheet" href="style.css">
8 <title>Proyecto Ecommerce</title>
9 </head>
10 <body>
11 <div class="container">
12 <header>
13 <div class="title">PRODUCT LIST</div>
14 <div class="icon-cart">
15 <!-- Referencia del icono https://flowbite.com/>
16 <svg class="w-6 h-6 text-gray-800 dark:text-white">
17 <path stroke="currentColor" stroke-linecap="round" stroke-width="1.5"></path>
18 </svg>
19 0
20 </div>
21 </header>
22
23 <div class="listProduct">
24 <div class="item">
25 <div class="image">
26
27 </div>
28 </div>
29 </div>
30
31 <div class="cartTab">
32 <h1>Shopping Cart</h1>
33 </div>
34 </div>
35 </body>
36 </html>
```

```
app.js
C:\Users\HP\Documents\Programacion\Semana 11> app.js ...
1 let iconCart = document.querySelector('.icon-cart');
2 let closeCart = document.querySelector('.close');
3 let body = document.querySelector('body');
4 let listProductsHTML = document.querySelector('.listProduct');
5 let listCartHTML = document.querySelector('.listCart');
6 let iconCartSpan = document.querySelector('.icon-cart span');
7 let derechaSpan = document.querySelector('.derecha span');
8
9 let listProduct = [];
10 let carts = [];
11
12 iconCart.addEventListener('click', () => {
13 body.classList.toggle('showCart');
14 })
15 closeCart.addEventListener('click', () => {
16 body.classList.toggle('showCart');
17 })
18
19
20 //Crear metodo para agregar los productos al html desde
21 const addDataToHTML = () => {
22 listProductsHTML.innerHTML = "";
23 if(listProducts.length > 0){
24 listProducts.forEach((product) => {
25 console.log(product.name);
26 let newProduct = document.createElement('div');
27 newProduct.classList.add('item');
28 newProduct.dataset.id = product.id;
29 newProduct.innerHTML = `` +
30 `<h2>${product.name}</h2>` +
31 `<div class="price">${product.price} MXP</div>`;
32 });
33 }
34 }
```



## Semana 12 – Estructuras2025

Carpeta: archivos-iniciales

### Propósito del módulo

Esta carpeta contiene los “archivos iniciales” que sirven como base para los ejercicios/prácticas de la Semana 12. Su función es proveer plantillas, estructuras o código de arranque sobre los cuales desarrollar las tareas de la semana — de modo que los estudiantes empiecen desde un entorno ya preparado, con configuraciones, defines, cabeceras o estructuras base listas para usar.

### Estructura del directorio (archivos-iniciales)

### Lenguaje(s) y formato esperados

- Lenguaje de programación: el utilizado en tu curso, posiblemente C o C++ .
- Uso de código fuente (`.c`, `.cpp`), cabeceras (`.h`), módulos de utilerías, configuración o defines comunes.
- La idea es entregar un “esqueleto” o un entorno base para que cada estudiante comience desde un punto unificado, evitando tener que reescribir estructuras o configuraciones comunes.

### Objetivos de la carpeta “archivos-iniciales”

Con esta carpeta se busca:

- Proporcionar un punto de partida común para todos los ejercicios de la semana — estructura básica, configuraciones, cabeceras, utilerías — de modo que los estudiantes se enfoquen en la lógica específica del ejercicio, no en detalles repetitivos.
- Asegurar consistencia entre ejercicios/prácticas: mismos defines, mismas cabeceras, mismas utilerías, formato unificado — lo que facilita corrección, comparación y mantenimiento.
- Reducir errores iniciales: al tener una base correcta ya validada, se minimiza el riesgo de errores de configuración, compilación o estructura, permitiendo concentrarse en la lógica del ejercicio.
- Agilizar la configuración del entorno: desde el primer momento, el estudiante tiene lo necesario para compilar, incluir cabeceras, reutilizar funciones comunes, etc.

### Descripción general del contenido

Dentro de esta carpeta sería habitual encontrar:

- Plantillas o archivos base para los ejercicios: por ejemplo archivos con función `main`, estructuras definidas, headers ya declarados, comentarios de guía, etc.
- Cabeceras comunes para definiciones compartidas: estructuras de datos comunes, macros, constantes, prototipos de funciones, defines de configuración, etc.
- Utilerías comunes que puedan ser reutilizadas en varios ejercicios: funciones de ayuda, manejo de errores, lectura/escritura, operaciones comunes, etc.
- Documentación mínima o guía: comentarios explicativos, instrucciones para compilar, estructura de carpetas y archivo base, notas sobre qué debe hacerse/modificarse en cada ejercicio.

## Análisis y recomendaciones

-  Uniformidad y consistencia : Tener archivos base común en cada ejercicio ayuda a mantener coherencia en todo el curso: estructura similar, nomenclatura uniforme, estilos consistentes. Esto simplifica revisión y retroalimentación.
-  Facilita enseñanza y aprendizaje : Los alumnos no pierden tiempo en configurar o crear cabeceras/base desde cero — pueden concentrarse en aprender lógica, algoritmos y estructura de datos.
-  Buena práctica de documentación : Es recomendable incluir comentarios claros, documentación sobre qué hace cada plantilla/base, instrucciones de compilación, y guía sobre qué modificar o completar. Esto mejora claridad, reproducibilidad y mantenimiento.
-  Reutilización de código : Utilerías comunes reducen duplicación de código, permiten aprovechar funciones útiles de forma uniforme y evitan inconsistencias entre ejercicios.
-  Mantenimiento cuidadoso : Si la carpeta base cambia (nombres, defines, cabeceras), puede afectar a muchos ejercicios — conviene documentar bien los cambios y avisar a quienes usen la base, para evitar conflictos.

## Conclusión

La carpeta `archivos-iniciales` en la Semana 12 del proyecto “Estructuras2025” sirve como fundamento: provee la estructura, configuración y utilerías necesarias para que los ejercicios de la semana inicien desde un entorno coherente, consistente y reutilizable. Implementar esta base ayuda a mantener uniformidad, facilitar el trabajo de los estudiantes y mejorar la organización global del curso.

Autoguardado

Archivo Inicio

index.html

```
> Users > HP > Documentos > Programación > Semana 12 > archivos-iniciales > index.html > ...
```

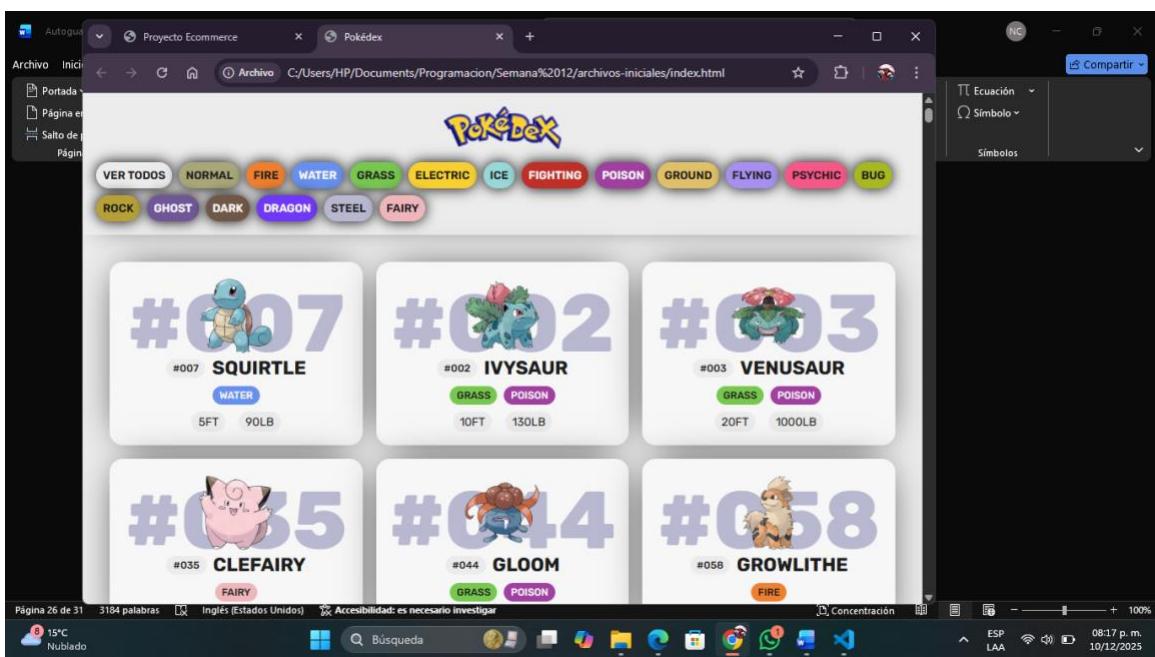
# main.css

```
1 import url('https://fonts.googleapis.com/css2?family=Rubik+Mono+One:wght@400;500;700;900;1000');
2
3 :root {
4 --clr-black: #1c1c1c;
5 --clr-gray: #e0e0e0;
6 --clr-white: #fff7f7f7;
7
8 --type-normal: #A8A878;
9 --type-fire: #F08030;
10 --type-water: #689900;
11 --type-grass: #70C850;
12 --type-electric: #8D0030;
13 --type-ice: #98D0B0;
14 --type-fighting: #C03028;
15 --type-poison: #A040A0;
16 --type-ground: #E0C068;
17 --type-flying: #A990F0;
18 --type-psychic: #F08088;
19 --type-bug: #B88B20;
20 --type-rock: #B8A038;
21 --type-ghost: #705998;
22 --type-dark: #705848;
23 --type-dragon: #7038F8;
24 --type-steel: #B8B8D0;
25 --type-fairy: #F0B0B0;
26 }
27
28 *
29 margin: 0;
30 padding: 0;
31 box-sizing: border-box;
32 color: var(--clr-black);
```

Página 26 de 31 15°C Nublado

Búsqueda

ESP LAA 08:16 p. m. 10/12/2025



## Semana 13 – Estructuras2025

### Propósito del módulo

Este directorio corresponde a la Semana 13 del curso “Estructuras2025”. En esta etapa el alumno debería consolidar y aplicar los conocimientos adquiridos anteriormente en ejercicios más elaborados, quizá combinando varias estructuras de datos, lógica, módulos auxiliares y buenas prácticas. El objetivo es avanzar hacia proyectos más completos o preparar la base para trabajos finales, integrando lo visto en las semanas anteriores.

### Estructura sugerida del directorio (Semana 13)

### Lenguaje(s) y formato esperados

- Lenguaje de programación: el que esté definido en el curso (e.g. C, C++, u otro).
- Uso de archivos fuente (.c, .cpp, etc.), cabeceras (.h), módulos auxiliares si aplica.
- Si el proyecto lo requiere: estructura modular, separación de lógica, utilerías comunes, posibles tests.
- Organización clara del código para facilitar mantenibilidad, comprensión y reutilización.

### Objetivos de la Semana 13

Al completar esta semana deberías:

- Implementar ejercicios o programas de complejidad media-alta o avanzada, combinando lo aprendido: estructuras de datos, funciones, modularidad, manejo de memoria (si aplica), entrada/salida, etc.
- Diseñar código estructurado, modular y limpio: dividir responsabilidades, usar utilerías comunes, comentar, documentar funciones, mantener un estilo coherente.
- Validar el funcionamiento mediante pruebas o casos de test, cubriendo distintos escenarios: valores normales, entradas límite, estructuras vacías o especiales, manejo de errores, etc.
- Preparar una base sólida para proyectos mayores o integraciones futuras del curso.
- Practicar buenas prácticas de desarrollo y documentación, facilitando la comprensión para ti mismo o posibles colaboradores.

### Contenido esperado / Descripción general

Posibles contenidos de esta semana:

- Ejercicios que combinen varias estructuras de datos o conceptos vistos anteriormente — por ejemplo, listas, nodos, estructuras dinámicas, algoritmos de manipulación, búsquedas, ordenamientos, gestión dinámica de memoria, etc.
- Módulos auxiliares consolidados: funciones utilitarias comunes, manejo de datos, utilerías de validación, logging, etc.
- Si usas pruebas: carpeta de tests con archivos de entrada/salida, scripts o instrucciones para ejecutar pruebas, verificar salida esperada y comportamiento.
- Documentación interna clara: comentarios en el código, cabeceras descriptivas, instrucciones de compilación/ejecución, qué hace cada módulo, posibles limitaciones o advertencias.
- Un README por módulo que describe propósito, estructura, instrucciones de uso — fundamental para mantener claridad y orden.

## Análisis y recomendaciones

-  Importancia de README por carpeta / módulo: Una descripción clara del contenido y propósito ayuda a cualquiera (incluyéndote a ti en el futuro) a entender rápidamente qué contiene cada parte del proyecto. De hecho, se recomienda crear un README en cada directorio principal de código. [¶1¶](#)
-  Modularidad y reutilización: Separar lógica en módulos / utilerías comunes mejora la mantenibilidad, facilita extensiones, y reduce duplicación de código.
-  Uso de pruebas / validaciones: Incluir casos de prueba ayuda a asegurar que tu código funciona correctamente bajo distintos escenarios, lo cual es especialmente útil en ejercicios complejos.
-  Documentar desde ahora: Si mantienes documentación consistente (README, comentarios, instrucciones), será mucho más sencillo revisar, compartir o retomar el proyecto más adelante. Markdown es ideal para ello. [¶2¶](#)
-  Consistencia en convenios: Usa nombres claros y uniformes para archivos, funciones y módulos (`ejercicio13\_.c/.cpp`, `utils13.`, `tests/`, etc.). Esto ayuda a la organización, colaboración y mantenimiento.
-  **Conclusión** La carpeta `Semana13` del curso “Estructuras2025”, usando esta plantilla, representa una etapa avanzada/intermedia-avanzada: con ejercicios más complejos, posible integración de varios conceptos, modularidad, pruebas y documentación clara. Si completas esta plantilla con tus archivos reales, tendrás un README ordenado, profesional y útil — tanto para ti como para quien revise o colabore en el proyecto.

```
C:\Users\HP\Documents>Programación>Semana13>index.html <--> database.md
```

```
<!DOCTYPE html><html lang="es"><head>
 <meta charset="UTF-8"/>
 <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no"/>
 <title>EduTrack - Sistema de Asistencia y Participación</title>
 <script src="https://cdn.tailwindcss.com"></script>
 <script src="https://d3js.cloudflare.com/ajax/libs/d3/5.16.0/d3.js"></script>
 <link rel="preconnect" href="https://fonts.googleapis.com">
 <link rel="preconnect" href="https://fonts.gstatic.com">
 <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap" rel="stylesheet">
```

```
<style>
 * { font-family: 'Inter', sans-serif; }
 .mono { font-family: 'JetBrains Mono', monospace; }
 .gradient-bg { background: linear-gradient(135deg, #1e73be, #3399cc); }
 .card-hover { transition: all 0.2s ease-out; }
 .card-hover:hover { transform: translateY(-2px); }
 .pulse-animation { animation: pulse 2s infinite; }
 @keyframes pulse { 0%, 100% { opacity: 1; } 50% { opacity: 0; } }
 .slide-in { animation: slideIn 0.3s ease-out; }
 @keyframes slideIn { from { transform: translateX(-100%); } to { transform: none; } }
 .loading-skeleton { background: linear-gradient(45deg, transparent, transparent 15px, #f0f0f0 15px, #f0f0f0 30px); }
 @keyframes loading { 0% { background-position: 0 0%; } 100% { background-position: 100% 0%; } }
```

```
</style>
<script src="https://statics.moonshot.cn/sdk/preview-wid">
</script>
<!-- Navigation -->
<nav class="bg-white shadow-sm border-b border-gray-200 py-2 px-4 lg:py-3 lg:px-6 lg:flex lg:justify-between items-center">
 <div class="flex items-center space-x-4">
 <div class="flex-shrink-0">
 <h1 class="text-2xl font-bold text-gray-900">
```

```
... database.md ...
```

```
C:\Users\HP\Documents>Programación>Semana13>database.md <--> Base de D
```

```
Estructura de Base de Datos (LocalStorage)
```

```
1. Tabla: students
... javascript
{
 id: "student_001",
 name: "Juan Pérez",
 email: "juan.perez@email.com",
 studentId: "2024001",
 classGroup: "10A",
 enrollmentDate: "2024-01-15",
 status: "active"
}
...
2. Tabla: classes
... javascript
{
 id: "class_001",
 name: "Matemáticas Avanzadas",
 subject: "MATH",
 teacherId: "teacher_001",
 schedule: "Lunes 8:00-10:00",
 semester: "2024-1",
 maxStudents: 30
}
...
3. Tabla: attendance_records
... javascript
```

The screenshot shows the EduTrack system interface. At the top, there's a navigation bar with links like 'Portada', 'Página en blanco', 'Salto de página', and 'Páginas'. The main title 'EduTrack Sistema de Asistencia y Participación' is displayed, along with a blue 'Exportar Datos' button. On the right, there's a sidebar with sections for 'Ecuación' and 'Símbolo', and a 'Símbolos' dropdown menu. The main content area features a 'Panel de Control' section with four cards: 'Total Estudiantes' (11), 'Asistencia Hoy' (0%), 'Participación Promedio' (0.0), and 'Clases Hoy' (0). Below this is a 'Clases' section listing four subjects with their respective times and student counts: 'Matemáticas Avanzadas' (Lunes 8:00-10:00, 5/30 students), 'Ciencias Naturales' (Martes 10:00-12:00, 6/25 students), 'Historia Universal' (Miércoles 14:00-16:00, 0 students), and 'Estructuras de Datos' (Miércoles 14:00-16:00, 0 students). A message from 'Kimi OK Computer' is visible in the bottom right of the classes section. The footer includes standard browser controls, a search bar, and system status indicators like battery level, temperature (15°C), and network connection.

## Semana 14 – Estructuras2025

### Propósito del módulo

Esta carpeta corresponde a la Semana 14 del curso “Estructuras2025”. En esta etapa se espera que el estudiante continúe consolidando los conocimientos previos, posiblemente abordando ejercicios más complejos o integrados — combinando estructuras de datos, algoritmos, módulos auxiliares, manejo de memoria o archivos (si aplica), y buenas prácticas de programación.

 Estructura sugerida del directorio (Semana 14)

### Lenguaje(s) y formato esperados

- Lenguaje de programación: el que se esté usando en el curso, por ejemplo C o C++ (u otro, según corresponda).
- Uso de archivos fuente (.c, .cpp, etc.), posiblemente con cabeceras (.h) si usas modularidad.
- Es posible que se empleen módulos auxiliares, utilerías comunes, carpetas de tests, o configuración adicional, según la complejidad de los ejercicios.
- Documentación en Markdown (.md) — ideal para README y otros documentos — por su simpleza, claridad y compatibilidad con plataformas como GitHub. 

### Objetivos de la Semana 14

Al completar esta semana, el estudiante debería poder:

- Integrar varios conceptos aprendidos previamente (estructuras de datos, algoritmos, modularidad, manejo de memoria/archivos, etc.) en ejercicios más elaborados.
- Escribir código modular y limpio: con funciones/módulos reutilizables, separación de responsabilidades, buenas prácticas de estilo y comentarios cuando sea necesario.
- (Opcional) Implementar pruebas o casos de test para verificar que el código funciona correctamente en distintos escenarios (entradas válidas, inválidas, casos límite, listas vacías, errores, etc.).
- Documentar adecuadamente el proyecto: qué hace cada archivo, cómo se compila/ejecuta, ejemplos de uso, posibles limitaciones o advertencias.
- Preparar una base sólida para posibles proyectos finales, integraciones o ampliaciones futuras del curso.

## Contenido esperado / Descripción general

Posibles contenidos de esta semana:

- Ejercicios o proyectos que combinen varias estructuras de datos y operaciones: manipulación dinámica de datos, estructuras complejas, algoritmos de búsqueda/ordenamiento/transformación, manejo de memoria o archivos.
- Módulos auxiliares consolidados: utilerías comunes, funciones reutilizables, abstracciones para facilitar manejo de estructuras, datos o memoria.
- Carpeta de pruebas (si decides incluirla): con archivos de entrada/salida o casos de test automáticos/manuales, para validar funcionalidad bajo distintos escenarios.
- Documentación clara: comentarios en el código, README por módulo, explicación de funciones/estructuras, instrucciones de compilación/ejecución, ejemplos de uso.

## Buenas prácticas y recomendaciones

-  Documentar desde el inicio: Un README al nivel de cada módulo (o semana) ayuda a que cualquiera —tú mismo en el futuro o un compañero— entienda rápidamente el proyecto. [¶3¶](#)
-  Modularidad y reutilización de código: Separar lógica en funciones/módulos auxiliares mejora la mantenibilidad, evita duplicación y facilita extensiones. [¶4¶](#)
-  Incluir pruebas / casos de test (si aplica): Ayuda a asegurar que el código responde correctamente bajo distintos escenarios, evita regresiones y facilita depuración. [¶5¶](#)
-  Comentarios y claridad del código: Los comentarios deben usarse para explicar “por qué” algo se hace — no sólo “qué hace” el código — y ayudar a mantenimiento futuro. [¶6¶](#)
-  Consistencia en nombres y estructura de carpetas: Usa convenciones claras para nombres de archivos, funciones y carpetas; esto facilita navegación, colaboración y automatización (por ejemplo, scripts de compilación o tests). [¶7¶](#)

## Conclusión

La carpeta `Semana14` del curso “Estructuras2025”, usando esta plantilla, puede representar una etapa avanzada: con ejercicios integrados, estructuración modular, documentación clara, posibles tests y un enfoque profesional de desarrollo. Si completas esta plantilla con tus archivos reales y sigues las buenas prácticas sugeridas, tendrás un proyecto organizado, mantenable y listo para compartir o continuar creciendo.

The screenshot shows a code editor with two tabs open: `index.html` and `app.js`.

**index.html:**

```

1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, i
6 <title>ToDo - Semana14</title>
7 <link rel="stylesheet" href="styles.css">
8 <meta name="description" content="Aplicación ToDo s
9 </head>
10 <body>
11 <header class="header">
12 <h1>Aplicación ToDo</h1>
13 <p class="subtitle">Registrar, consultar, editar
14 </header>
15
16 <main class="container">
17 <section class="panel">
18 <h2>Registrar tarea</h2>
19 <form id="task-form" class="form">
20 <label for="title">Título</label>
21 <input type="text" id="title" placeholder="<
22
23 <label for="description">Descripción</la
24 <textarea id="description" placeholder="<
25
26 <div class="form-actions">
27 <button type="submit" class="btn">Regis
28 <button type="button" id="clear-btn" c
29 </div>
30 </form>
31 </section>
32 </main>

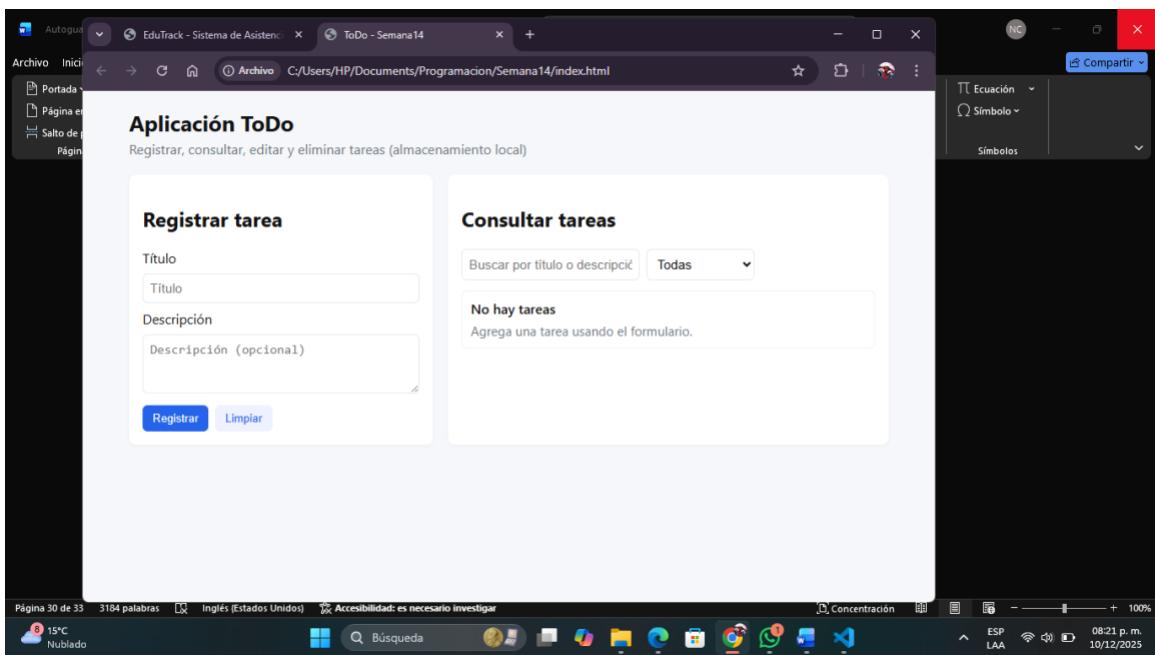
```

**app.js:**

```

C:\Users\HP\Documents\Programación\Semana14\apps\hana14> JS app.js ...
1 // app.js - funcionalidad CRUD usando localStorage
2 (function(){
3 const STORAGE_KEY = 'todo_semana14_tasks_v1';
4 let tasks = [];
5
6 // elementos
7 const taskForm = document.getElementById('task-form');
8 const titleInput = document.getElementById('title');
9 const descInput = document.getElementById('description');
10 const clearBtn = document.getElementById('clear-btn');
11
12 const searchInput = document.getElementById('search');
13 const filterSelect = document.getElementById('filter');
14 const taskList = document.getElementById('task-list');
15
16 const editModal = document.getElementById('edit-modal');
17 const editForm = document.getElementById('edit-form');
18 const editIdInput = document.getElementById('edit-id');
19 const editTitleInput = document.getElementById('edit-t');
20 const editDescInput = document.getElementById('edit-de');
21 const editDoneInput = document.getElementById('edit-d');
22 const editCancelBtn = document.getElementById('edit-ca
23
24 // init
25 loadTasks();
26 bindEvents();
27 renderTasks();
28
29 function loadTasks(){
30 try{
31 const raw = localStorage.getItem(STORAGE_KEY);
32 tasks = raw ? JSON.parse(raw) : [];

```



## Semana 15 – Estructuras2025

### Propósito del módulo

Este directorio corresponde a la Semana 15 del curso “Estructuras2025”. En esta etapa se asume que el estudiante ya cuenta con conocimientos previos consolidados, por lo que los ejercicios o trabajos de esta semana deben integrar conceptos, estructuras y prácticas avanzadas; preparar al estudiante para retos finales, proyectos integradores, o consolidación de competencias del curso.

#### Estructura sugerida del directorio (Semana 15)

## Lenguaje(s) y formato esperados

- Lenguaje de programación: el que utilice el curso (por ejemplo C, C++, otro — según corresponda).
- Archivos fuente (`.c`, `.cpp`, etc.), posibles archivos de cabecera (`.h`), módulos auxiliares, librerías internas, etc.
- Opcionalmente, una estructura modular: utilerías compartidas, separación de lógica, y si se requiere: carpeta de pruebas.
- Formato de documentación en Markdown (`.md`) para README (y posiblemente otros documentos), facilitando lectura en GitHub.

## Objetivos de la Semana 15

Al completar esta semana, se espera que el estudiante:

- Integre varias estructuras de datos y conceptos: combinación de estructuras, algoritmos, modularidad, manejo de memoria o datos, operaciones complejas, etc.
- Diseñe código bien estructurado: modular, reutilizable, claro, con separación lógica de responsabilidades y utilizando funciones/módulos auxiliares si es necesario.
- Si aplica: implemente pruebas o casos de test que permitan validar el funcionamiento del código bajo distintos escenarios — entradas normales, límites, errores, casos extremos.
- Documente adecuadamente su trabajo: código comentado cuando necesario, explicación de funciones/estructuras, instrucciones de compilación y uso, ejemplos simples de ejecución.

- Prepárese para trabajos finales, integración con otros módulos, o uso del contenido aprendido en proyectos más grandes o más complejos.

## Contenido esperado / Descripción general

Durante esta semana 15 podrían incluirse actividades como:

- Ejercicios o proyectos que combinan varias estructuras de datos: listas, árboles, colas, pilas, estructuras dinámicas, etc.
- Algoritmos más complejos: búsquedas, ordenamientos, manipulaciones, operaciones sobre estructuras, manejo de memoria dinámica (si aplica), validación de datos, entrada/salida, etc.
- Módulos auxiliares utilitarios: funciones genéricas, utilerías comunes, manejo de errores, gestión de memoria, abstracciones.
- Casos de prueba — si se decide incluirlos —: entradas/salidas típicas, pruebas de comportamiento en diferentes casos, validaciones, manejo de errores.
- Documentación clara: tanto del código como de la estructura del proyecto, instrucciones de compilación/ejecución, posibles advertencias o notas especiales.

## Buenas prácticas y recomendaciones

-  Utiliza README en cada módulo/directorio : ayuda a que cualquiera (tú en el futuro o un compañero) entienda rápidamente qué contiene y cuál es el propósito. [🔗 0](#)
-  Modularidad y reutilización : separar lógica en funciones/modulos reutilizables mejora mantenibilidad, evita duplicación de código, y facilita extensiones o integraciones futuras. [🔗 1](#)
-  Incluye pruebas o validaciones (si aplica) : ayuda a asegurar que tu código funcione correctamente en distintos escenarios, y facilita detectar errores al modificar o extender el código. [🔗 2](#)
-  Documentación clara y actualizada : usar Markdown (README, comentarios, guías) permite que tu proyecto sea legible, usable, compatible y fácil de mantener. [🔗 3](#)
-  Convenciones consistentes : usar nombres claros y uniformes de archivos, funciones, carpetas, pruebas — facilita navegación, colaboración y mantenimiento del repositorio.

## Conclusión

La carpeta `Semana15` del curso “Estructuras2025”, siguiendo esta plantilla, representa un nivel avanzado: con posibles proyectos integradores, estructuras compuestas, modularidad, pruebas y documentación adecuada. Al completar esta plantilla con tus archivos reales y adoptar buenas prácticas, tendrás un módulo bien estructurado, claro, mantenable y profesional — ideal para continuar con el proyecto, expandirlo o compartirlo.

```

JS graph.js 5 × JS code.js
C:\Users\HP\Documents\Programacion\Semana15\graph> JS graph.js > ...
1 /*
2 2 - 0
3 / \
4 1 - 3
5 */
6
7
8 //Edge list
9
10 const graph = [
11 [0, 2],
12 [2, 3],
13 [2, 1],
14 [1, 3]
15];
16
17 // Adjacent List
18 // 0 1 2 3
19 const graph = [[2], [2, 3], [0, 1, 3], [1, 2]]
20
21 const graph = {
22 0: [2],
23 1: [2, 3],
24 2: [0, 1, 3],
25 3: [1, 2]
26 }
27
28 //Adjacent Matrix
29
30 const graph = [
31 [0, 0, 1, 0],
32 [0, 0, 1, 1],

```

```

JS graph_2.js × JS doble.js
C:\Users\HP\Documents\Programacion\Semana15\graph> JS graph_2.js > ...
1 class Graph {
2 Click to add a breakpoint `() {
3 this.nodes = 0;
4 this.adjacentList = {};
5 }
6
7 addVertex(node) {
8 this.adjacentList[node] = [];
9 this.nodes++;
10 }
11
12 addEdge(node1, node2) {
13 this.adjacentList[node1].push(node2);
14 this.adjacentList[node2].push(node1);
15 }
16
17 }
18
19 const myGraph = new Graph();

```

Página 32 de 1 ⑤ 0 100% 15°C Nublado Búsqueda ESP LAA 08:23 p. m. 10/12/2025

```

JS graph.js 5 × JS code.js
C:\Users\HP\Documents\Programacion\Semana15\graph> JS code.js > ...
1 /*
2 //Declaracion de variable global
3 let x = 10;
4
5 function incrementX() {
6 let y = 100; // Variable local
7 x += 1;
8 }
9
10 console.log("Before increment:", x);
11 incrementX();
12
13 console.log("After increment:", x);
14 determinaValor(10);
15
16 determinaValor();
17
18 // Intentando acceder a la variable local y fuera de su
19 //console.log("Accessing local variable y:", y);
20
21 function determinaValor(num){
22 let mensaje;
23 if(num > 0 && num <= 10){
24 mensaje = "El número es una unidad";
25 }else if(num > 10 && num <= 100){
26 mensaje = "El número es una decena";
27 }else if(num > 100 && num <= 1000){
28 mensaje = "El número es una centena";
29 }else if(num > 1000 && num <= 10000){
30 mensaje = "El número es un millar";
31 }else if(num > 10000 && num <= 100000){
32 mensaje = "El número es una decena de millar";

```

```

JS graph_2.js × JS doble.js
C:\Users\HP\Documents\Programacion\Semana15\graph> JS doble.js > ...
1 class Node {
2 constructor(value) {
3 this.value = value;
4 this.next = null;
5 this.prev = null;
6 }
7
8 class MyDoublyLinkedList {
9 constructor(value) {
10 this.head = {
11 value: value,
12 next: null,
13 prev: null
14 };
15 this.tail = this.head;
16
17 this.length = 1;
18 }
19
20 append(value) {
21 const newNode = new Node(value);
22 newNode.prev = this.tail;
23 this.tail.next = newNode;
24 this.tail = newNode;
25 this.length++;
26 return this;
27 }
28
29 prepend(value) {
30 const newNode = new Node(value);
31 newNode.next = this.head;
32 this.head.prev = newNode;

```

Página 32 de 1 ⑤ 0 100% 15°C Nublado Búsqueda ESP LAA 08:24 p. m. 10/12/2025

## Semana 16 – Estructuras2025

### Propósito del módulo

Este directorio corresponde a la Semana 16 del curso “Estructuras2025”. En esta fase final o avanzada, se espera que los estudiantes integren conocimientos previos — estructuras de datos, algoritmos, modularidad, manejo de memoria/archivos — para implementar ejercicios o proyectos más completos, consolidados y bien estructurados.

### Estructura sugerida del directorio (Semana 16)

### Lenguaje(s) y formato esperados

- Lenguaje de programación: el que se utilice en el curso — por ejemplo C, C++, u otro.
- Archivos fuente (.c, .cpp, etc.), posiblemente con cabeceras (.h) si usas modularidad.
- Si corresponde: módulos auxiliares (para funciones comunes), utilerías compartidas, separación de lógica, etc.
- Si decides implementar pruebas: carpeta `tests/` con casos de prueba, entrada/salida, validación de resultados.
- Documentación en Markdown (.md) para README (y otros documentos si decides expandirla).

### Objetivos de la Semana 16

Al completar esta semana, los objetivos deberían ser:

- Integrar varios conceptos del curso: estructuras de datos, algoritmos, modularidad, manejo de memoria o datos, operaciones complejas.
- Construir soluciones completas y funcionales: programas o módulos que funcionen de forma robusta, con código limpio, modular y mantenable.
- Si aplica, implementar pruebas o casos de test para asegurar que el código funcione correctamente en distintos escenarios (entradas normales, límites, errores, estructuras vacías, etc.).
- Documentar claramente: comentar funciones/estructuras, explicar qué hace cada archivo, proveer instrucciones de compilación/ejecución, ejemplos de uso si aplica.

- Mantener buenas prácticas de programación: estilos de código legibles, nombres claros, modularidad, organización coherente — esto facilita mantenimiento, revisión o colaboración.

## Contenido esperado / Descripción general

Algunos ejemplos de lo que esta semana podría implicar:

- Ejercicios o proyectos que combinen múltiples estructuras de datos o funcionalidades complejas — por ejemplo: listas, árboles, colas, pilas, operaciones sobre datos dinámicos, manejo de memoria, lectura/escritura, etc.
- Módulos auxiliares reutilizables: funciones de utilidad, manejo de estructuras, manejo de errores, gestión de memoria, abstracciones comunes.
- Si usas pruebas: escenarios variados, entradas/salidas, validación de resultados, casos límite, para asegurar robustez.
- Código limpio y bien organizado: separación de responsabilidades, estructura clara de carpetas, nombres descriptivos, comentarios bien ubicados.
- Documentación suficiente para cualquiera — tú en el futuro o compañeros — pueda entender, compilar y usar el código fácilmente.

## Buenas prácticas y recomendaciones

-  Incluye siempre un README por módulo/semana: ayuda a entender qué contiene, para qué sirve, cómo compilar/usrar. Un buen README mejora la mantenibilidad del proyecto. [¶1¶](#)
-  Mantén una estructura de carpetas clara y coherente: facilita navegación, entendimiento, mantenimiento y escalabilidad del proyecto. [¶2¶](#)
-  Usa nombres descriptivos para archivos, funciones y módulos: mejora la legibilidad y claridad del código. [¶3¶](#)
-  Si implementas pruebas — especialmente en proyectos complejos — documenta cómo ejecutarlas y qué cubren, para asegurar la funcionalidad bajo distintos escenarios. [¶4¶](#)
-  Documenta tu código internamente: usa comentarios cuando sea necesario, explica decisiones importantes, y describe qué hace cada módulo/función. Esto facilita que tú o alguien más lo entienda en el futuro. [¶5¶](#)

## Conclusión

La carpeta 'Semana16` del curso "Estructuras2025", con esta plantilla, representa una etapa avanzada — con ejercicios más elaborados, posible integración de varios conceptos, modularidad, pruebas y documentación. Si completas la plantilla con tus archivos reales y adoptas buenas prácticas, tendrás un módulo bien estructurado, claro, mantenible y profesional — ideal para continuar expandiendo el proyecto, revisarlo en el futuro o compartirlo con otros.

```

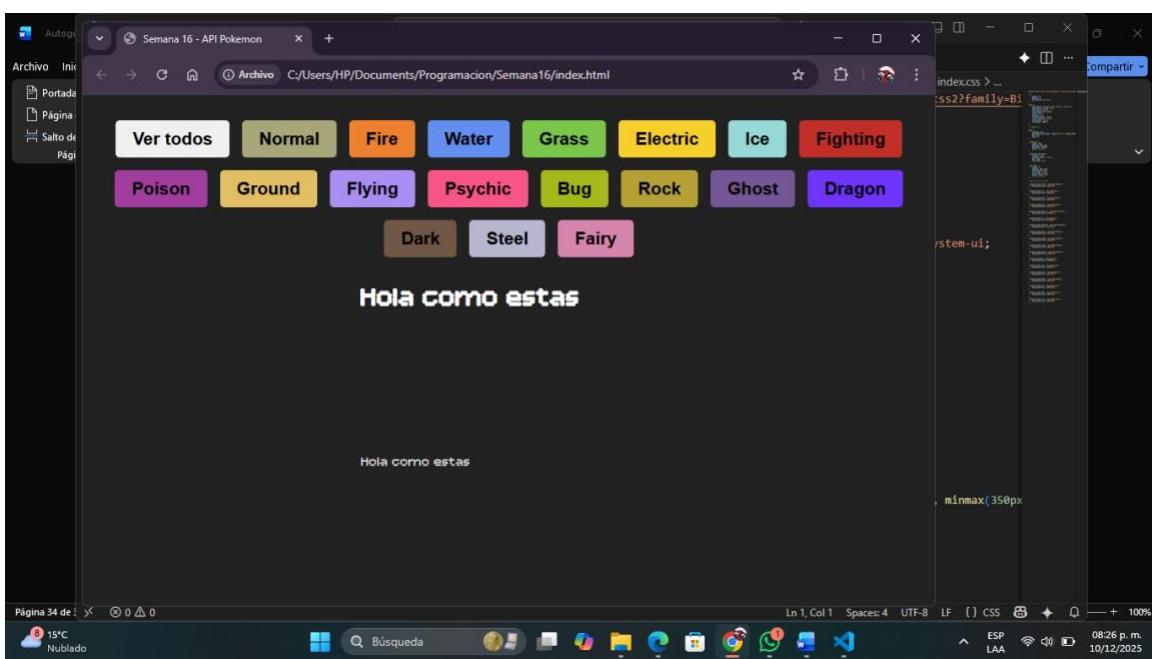
index.html
C:\Users\HP\Documents\Programacion\Semana16> index.html > ...
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, i
6 <title>Semana 16 - API Pokemon</title>
7 <link rel="stylesheet" href="index.css">
8 </head>
9 <body>
10 <header class="container">
11 <nav class="container-btn-types">
12 <button class="btn-types">Ver todos</button>
13 <button class="btn-types btn-type-normal">Normal</button>
14 <button class="btn-types btn-type-fire">Fire</button>
15 <button class="btn-types btn-type-water">Water</button>
16 <button class="btn-types btn-type-grass">Grass</button>
17 <button class="btn-types btn-type-electric">Electric</button>
18 <button class="btn-types btn-type-ice">Ice</button>
19 <button class="btn-types btn-type-fighting">Fighting</button>
20 <button class="btn-types btn-type-poison">Poison</button>
21 <button class="btn-types btn-type-ground">Ground</button>
22 <button class="btn-types btn-type-flying">Flying</button>
23 <button class="btn-types btn-type-psychic">Psychic</button>
24 <button class="btn-types btn-type-bug">Bug</button>
25 <button class="btn-types btn-type-rock">Rock</button>
26 <button class="btn-types btn-type-ghost">Ghost</button>
27 <button class="btn-types btn-type-dragon">Dragon</button>
28 <button class="btn-types btn-type-dark">Dark</button>
29 <button class="btn-types btn-type-steel">Steel</button>
30 <button class="btn-types btn-type-fairy">Fairy</button>
31 </nav>
32 </header>

```

```

index.css
C:\Users\HP\Documents\Programacion\Semana16> # index.css > ...
1 import url('https://fonts.googleapis.com/css2?family=Bitcount+Prop+Single:latin');
2 * {
3 margin: 0;
4 padding: 0;
5 box-sizing: border-box;
6 }
7
8 body {
9 font-family: "Bitcount Prop Single", system-ui;
10 font-optical-sizing: auto;
11 font-style: normal;
12 background-color: #f0f0f0;
13 color: #000000;
14 display: flex;
15 flex-direction: column;
16 justify-content: center;
17 align-items: center;
18 min-height: 100vh;
19 }
20
21 /* Imagenes */
22
23 main {
24 display: grid;
25 grid-template-columns: repeat(auto-fit, minmax(350px
26 gap: 5px;
27 flex: 1;
28 }
29
30 .container {
31 width: 100%;
32 }

```



# Insignia Del Curso de Java

<https://developers.google.com/profile/u/108903504084318177919>

9:58

Google for Developers

Insignias favoritas

Aprendizaje

Actividades de aprendizaje completadas en el ecosistema de desarrolladores de Google

Aprendizaje

[Learn JavaScript](#) 15 dic 2025

Detalles de la insignia

Compartir X f in

developer.google.com

9:59

Google for Developers

Insignias favoritas

Aprendizaje

Actividades de aprendizaje completadas en el ecosistema de desarrolladores de Google

Compartir X f in

✓ ¡Ya tienes esta insignia!

Aprendizaje

[Learn JavaScript](#) 15 dic 2025

Política de contenido

< [opers.google.com](#) ⌂ ...

Google for Developers

