Recetario Digital

Proyecto final - Entrega Final

Axel Guillermo Martinez Martinez
Correo: axel.martinez5005@alumnos.udg.mx
Código: 220450053
Materia: Estructuras de Datos
Profesor: Alfredo Gutierrez Hernandez
NRC: 200219
Clave de Materia: IL354
Fecha de Entrega: 21/11/2025

1. Autoevaluación

Concepto	Sí	No	Acumulación
Bajé el trabajo de internet o alguien me lo pasó (aunque sea de forma parcial)	-100 pts	0 pts	0 pts
Incluí el código fuente en formato de texto (sólo si funciona cumpliento todos los requerimientos)	+25 pts	0 pts	25 pts
Incluí las impresiones de pantalla (sólo si funciona cumpliento todos los requerimientos)	+25 pts	0 pts	25 pts
Incluí una portada que identifica mi trabajo (nombre, código, materia, fecha, título)	+25 pts	0 pts	25 pts
Incluí una descripción y conclusiones de mi trabajo	+25 pts	0 pts	25 pts
Suma:			100 pts

Índice

Introducción

El proyecto del recetario surgió con la idea de desarrollar un sistema que permita administrar recetas de manera ordenada, eficiente y flexible, facilitando tareas como agregar, mostrar, buscar, editar y eliminar recetas. Desde el inicio se buscó crear una herramienta sencilla para el usuario, pero con una estructura bien pensada que refleja el manejo de datos dinámicos.

Desde el inicio se decidió trabajar con estructuras dinámicas hechas a mano, específicamente listas simplemente ligadas y doblemente ligadas, para tener un control total sobre la forma en que se almacenan y manipulan los datos. Cada receta maneja internamente su propia lista de ingredientes, y todas las recetas se organizan dentro de un recetario general que permite realizar operaciones como agregar, buscar, editar, mostrar o eliminar.

También se integró la capacidad de guardar la información en archivos y cargarla nuevamente cuando el programa se ejecuta, asegurando la persistencia de los datos y permitiendo que el recetario se mantenga aunque el sistema se cierre. Todo el proyecto se construyó separando responsabilidades en clases específicas y manteniendo una estructura clara y lógica.

En resumen, este recetario combina una estructura interna flexible pues al utilizar a la memoria dinámica ya no es necesario el uso de arreglos estáticos fijos pudiendo crecer la información dentro del recetario, también se cuenta con un manejo manual de memoria y una interfaz sencilla para el usuario, con el objetivo de ofrecer un sistema funcional que demuestre el uso correcto de listas dinámicas.

1. Requerimientos funcionales

Agregar receta.
El sistema permitirá ingresar una receta completa (nombre, categoría, autor, tiempo en minutos, procedimiento) y su lista de ingredientes (nombre, unidad, cantidad).

Mostrar recetas.
Mostrar todas las recetas en orden de la lista (recorrido desde el primer nodo hasta el último).

Mostrar resumen
(nombre, categoría, autor, tiempo) o detalle en la búsqueda (incluye procedimiento e ingredientes).

Buscar recetas.
Al buscar una receta se muestran todos los datos de esta misma por cuestión de diseño.

- Buscar por nombre: retorna la primera coincidencia exacta.
- Buscar por categoría: lista recetas pertenecientes a la categoría solicitada.

Editar receta.

- Permitir editar campos principales (nombre, categoría, autor, tiempo, procedimiento).
- Reemplazar la lista completa de ingredientes.

Eliminar receta.

- Eliminar una receta por nombre.
- Eliminar todas las recetas (vaciar recetario).

Gestión de ingredientes. (dentro de una receta)

- Agregar ingrediente (inserción ordenada por nombre dentro de la lista de ingredientes de la receta).
- Eliminar ingrediente por nombre.
- Modificar cantidad de ingredientes.
- Eliminar todos los ingredientes de una receta.

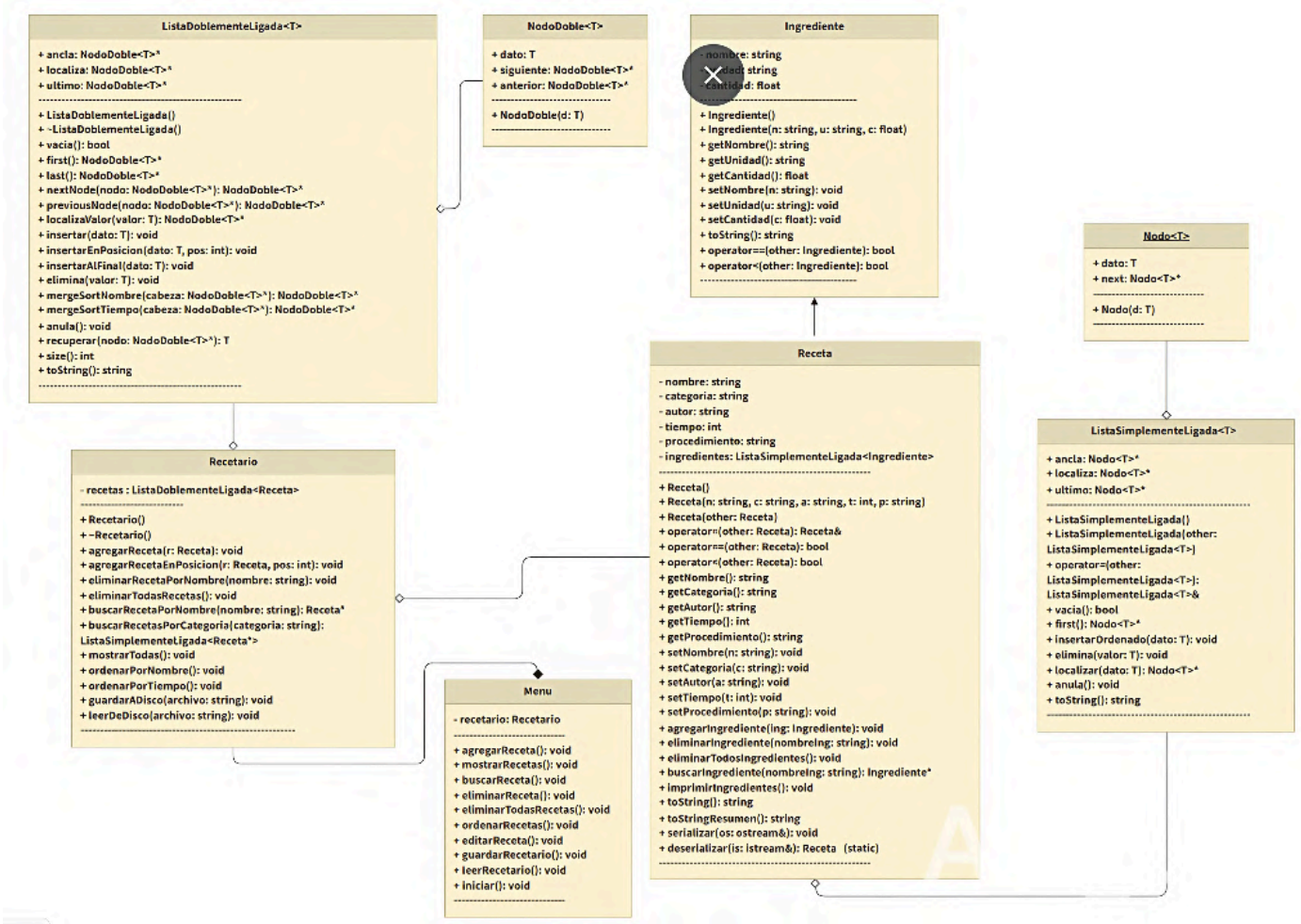
Almacenamiento.

- Guardar todas las recetas a disco en Recetas.txt (formato delimitado especificado).
- Leer desde Recetas.txt y agregar las recetas leídas sin borrar las ya cargadas en memoria. Mantener el orden del archivo.

Ordenamiento.
Ordenar recetas por nombre o por tiempo de preparación (merge sort implementado sobre la lista doblemente ligada).

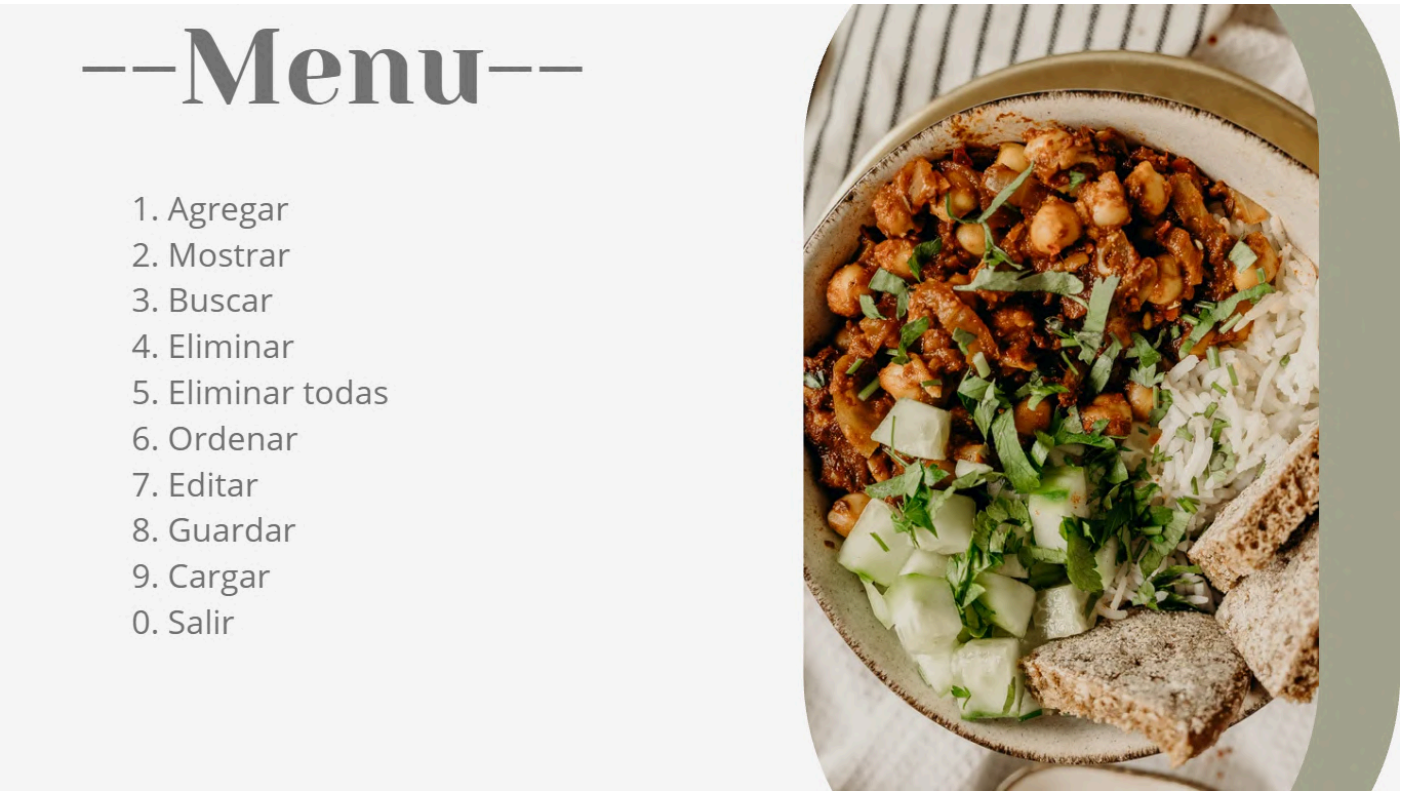
Pruebas: Se pondrá a prueba con 10 recetas.

2.1 UML


ListaDoblementeLigada<T>
+ ancla: NodoDoble<T>*
+ localiza: NodoDoble<T>*
+ ultimo: NodoDoble<T>*
+ ListaDoblementeLigada()
+ ~ListaDoblementeLigada()
+ vacia(): bool
+ first(): NodoDoble<T>*
+ last(): NodoDoble<T>*
+ nextNode(nodo: NodoDoble<T>*): NodoDoble<T>*
+ previousNode(nodo: NodoDoble<T>*): NodoDoble<T>*
+ localizaValor(valor: T): NodoDoble<T>*
+ insertar(dato: T): void
+ insertarEnPosicion(dato: T, pos: int): void
+ insertarAlFinal(dato: T): void
+ elimina(valor: T): void
+ mergeSortNombre(cabeza: NodoDoble<T>*): NodoDoble<T>*
+ mergeSortTiempo(cabeza: NodoDoble<T>*): NodoDoble<T>*
+ anula(): void
+ recuperar(nodo: NodoDoble<T>*): T
+ size(): int
+ toString(): string
NodoDoble<T>
+ dato: T
+ siguiente: NodoDoble<T>*
+ anterior: NodoDoble<T>*
+ NodoDoble(d: T)
Ingrediente
- nombre: string
- unidad: string
- cantidad: float
+ Ingrediente()
+ Ingrediente(n: string, u: string, c: float)
+ getNombre(): string
+ getUnidad(): string
+ getCantidad(): float
+ setNombre(n: string): void
+ setUnidad(u: string): void
+ setCantidad(c: float): void
+ toString(): string
+ operator==(other: Ingrediente): bool
+ operator<(other: Ingrediente): bool
Nodo<T>
+ dato: T
+ next: Nodo<T>*
+ Nodo(d: T)
Receta
- nombre: string
- categoria: string
- autor: string
- tiempo: int
- procedimiento: string
- ingredientes: ListaSimplementeLigada<Ingrediente>
+ Receta()
+ Receta(n: string, c: string, a: string, t: int, p: string)
+ Receta(other: Receta)
+ operator=(other: Receta): Receta&
+ operator==(other: Receta): bool
+ operator<(other: Receta): bool
+ getNombre(): string
+ getCategoria(): string
+ getAutor(): string
+ getTiempo(): int
+ getProcedimiento(): string
+ setNombre(n: string): void
+ setCategoria(c: string): void
+ setAutor(a: string): void
+ setTiempo(t: int): void
+ setProcedimiento(p: string): void
+ agregarIngrediente(ing: Ingrediente): void
+ eliminarIngrediente(nombreIng: string): void
+ eliminarTodosIngredientes(): void
+ buscarIngrediente(nombreIng: string): Ingrediente*
+ imprimirIngredientes(): void
+ toString(): string
+ toStringResumen(): string
+ serializar(os: ostream&): void
+ deserializar(is: istream&): Receta (static)
ListaSimplementeLigada<T>
+ ancla: Nodo<T>*
+ localiza: Nodo<T>*
+ ultimo: Nodo<T>*
+ ListaSimplementeLigada()
+ ListaSimplementeLigada(other: ListaSimplementeLigada<T>)
+ operator=(other: ListaSimplementeLigada<T>): ListaSimplementeLigada<T>&
+ vacia(): bool
+ first(): Nodo<T>*
+ insertarOrdenado(dato: T): void
+ elimina(valor: T): void
+ localizar(dato: T): Nodo<T>*
+ anula(): void
+ toString(): string
Recetario
- recetas : ListaDoblementeLigada<Receta>
+ Recetario()
+ ~Recetario()
+ agregarReceta(r: Receta): void
+ agregarRecetaEnPosicion(r: Receta, pos: int): void
+ eliminarRecetaPorNombre(nombre: string): void
+ eliminarTodasRecetas(): void
+ buscarRecetaPorNombre(nombre: string): Receta*
+ buscarRecetasPorCategoria(categoria: string): ListaSimplementeLigada<Receta*>
+ mostrarTodas(): void
+ ordenarPorNombre(): void
+ ordenarPorTiempo(): void
+ guardarADisco(archivo: string): void
+ leerDeDisco(archivo: string): void
Menu
- recetario: Recetario
+ agregarReceta(): void
+ mostrarRecetas(): void
+ buscarReceta(): void
+ eliminarReceta(): void
+ eliminarTodasRecetas(): void
+ ordenarRecetas(): void
+ editarReceta(): void
+ guardarRecetario(): void
+ leerRecetario(): void
+ iniciar(): void

2.2 Diseño de interfaces

Menú



Flujo desde aquí:

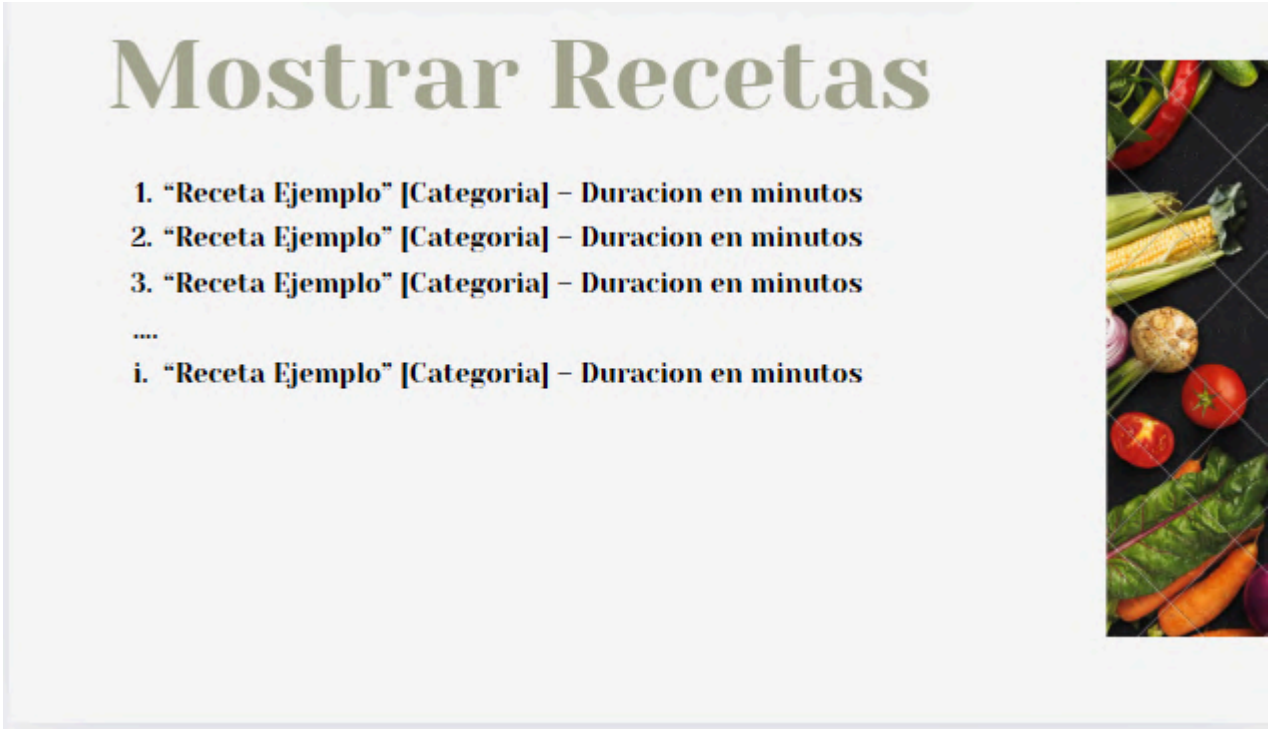
- 1 → Agregar Receta
- 2 → Mostrar Recetas
- 3 → Buscar Receta
- 4 → Eliminar Receta
- 5 → Eliminar Todas
- 6 → Ordenar Recetas
- 7 → Editar Receta
- 8 → Guardar Recetario
- 9 → Cargar Recetario

Agregar Receta

Flujo:

- Después de agregar → Menú Principal

Mostrar Recetas



- Después de mostrar listado→ Menú Principal

Buscar Receta

- Después de buscar receta→ Menú Principal

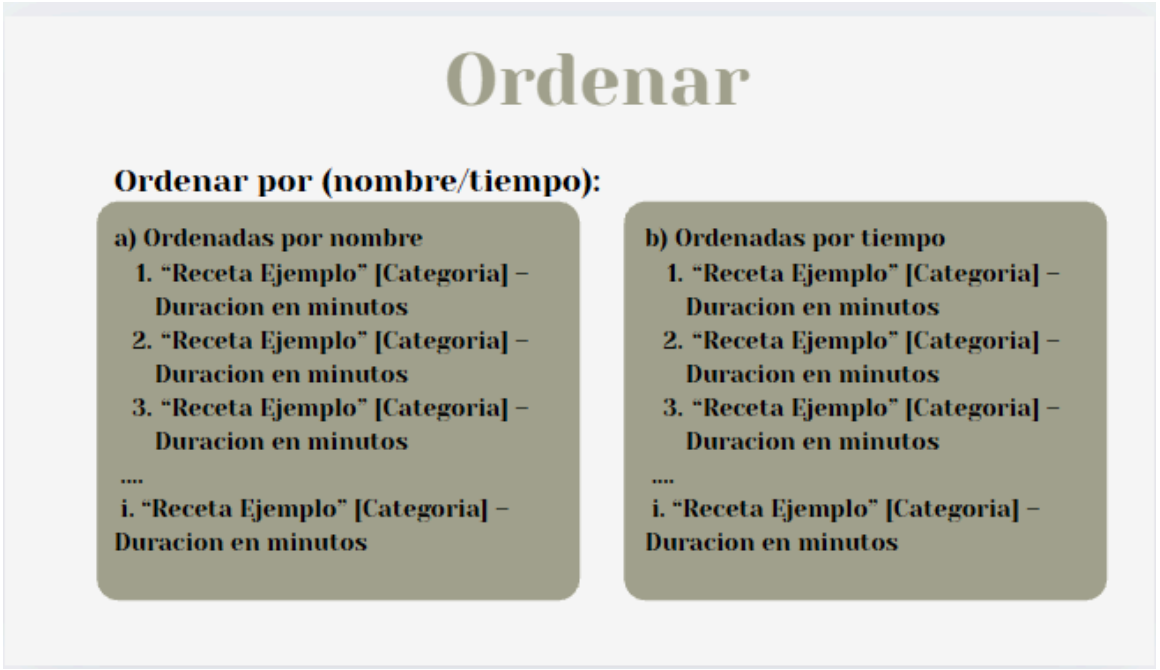
Eliminar Recetas

- Después de eliminar receta→ Menú Principal

Eliminar todas las recetas

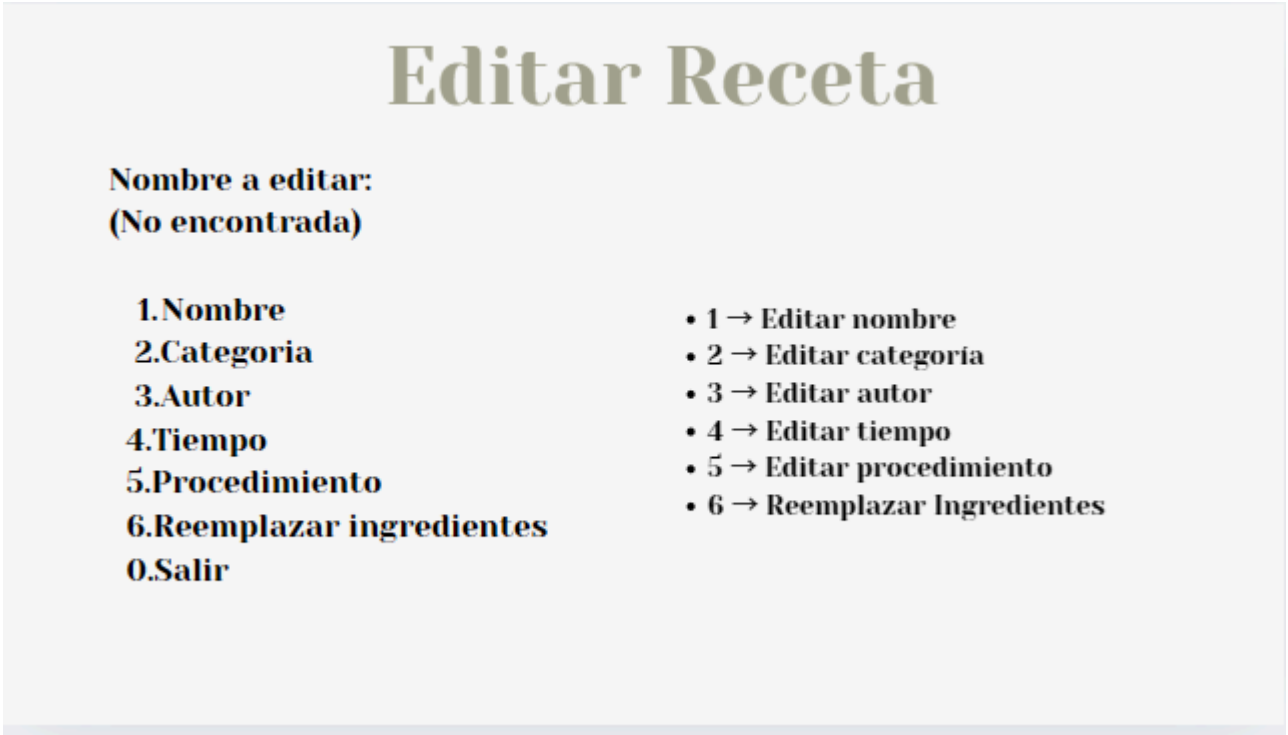
- Después de eliminar todas→ Menú Principal

Ordenar Recetas



- Después de ordenar→ Menú Principal

Editar Receta



1 → Editar nombre

2 → Editar categoría

3 → Editar autor

4 → Editar tiempo

5 → Editar procedimiento

6 → Reemplazar Ingredientes

0→ Menú Principal

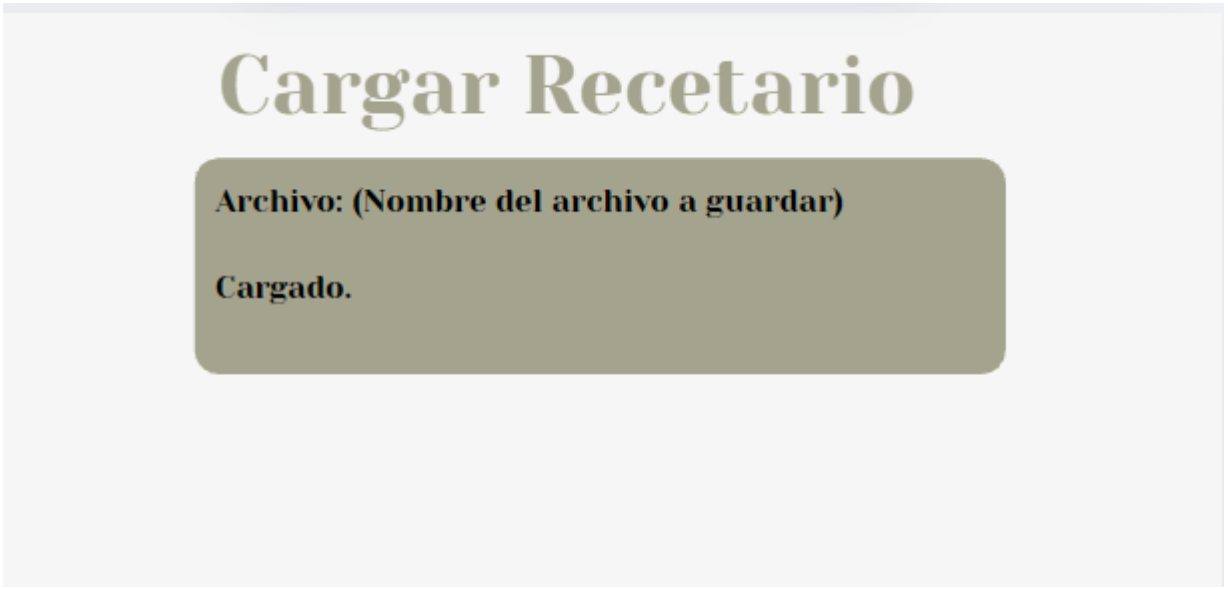
Reemplazar Ingredientes

- Finaliza → vuelve a Editar Receta

Guardar al disco

- Después → Menú Principal

Cargar al disco



- Después → Menú Principal

3. Codificación

1. main.cpp

```cpp
#include "Menu.hpp"

int main(){
    Menu menu;
    menu.iniciar();
    return 0;
}
```

2. menu.hpp

```cpp
#ifndef MENU_HPP
#define MENU_HPP

#include "Recetario.hpp"
#include <string>

class Menu {
private:
    Recetario recetario;
public:
    void agregarReceta();
    void mostrarRecetas();
    void buscarReceta();
    void eliminarReceta();
    void eliminarTodasRecetas();
    void ordenarRecetas();
    void editarReceta();
    void guardarRecetario();
    void leerRecetario();
    void iniciar();
};

#endif
```

3. menu.cpp:

```
#include "Menu.hpp"
#include <iostream>
#include <string>
#include <stdexcept>

void Menu::agregarReceta() {
    try {
        std::string nombre, categoria, autor, procedimiento;
        int tiempo, nIng;

        std::cin.ignore();
        std::cout << "Nombre: ";
        std::getline(std::cin, nombre);

        std::cout << "Categoria: ";
        std::getline(std::cin, categoria);

        std::cout << "Autor: ";
        std::getline(std::cin, autor);

        std::cout << "Tiempo (min): ";
        std::cin >> tiempo;
        std::cin.ignore();

        std::cout << "Procedimiento: ";
        std::getline(std::cin, procedimiento);

        Receta r(nombre, categoria, autor, tiempo, procedimiento);

        std::cout << "Numero de ingredientes: ";
        std::cin >> nIng;
        std::cin.ignore();

        for (int i = 0; i < nIng; i++) {
            std::string n, u;
            float c;

            std::cout << "Ingrediente " << i + 1 << " nombre: ";
            std::getline(std::cin, n);

            std::cout << "Unidad: ";
```

```
            std::getline(std::cin, u);

            std::cout << "Cantidad: ";
            std::cin >> c;
            std::cin.ignore();

            r.agregarIngrediente(Ingrediente(n, u, c));
        }

        recetario.mostrarTodas();
        std::cout << "Posicion a insertar (0 principio): ";
        int pos;
        std::cin >> pos;
        std::cin.ignore();

        recetario.agregarRecetaEnPosicion(r, pos);

        std::cout << "Receta agregada correctamente.\n";

    } catch (const std::exception& e) {
        std::cout << "Error: " << e.what() << "\n";
    }
}


void Menu::mostrarRecetas() {
    recetario.mostrarTodas();
}

void Menu::buscarReceta() {
    try {
        std::string opcion, valor;
        std::cin.ignore();

        std::cout << "Buscar por (nombre/categoria): ";
        std::getline(std::cin, opcion);

        if (opcion == "nombre") {
            std::cout << "Nombre: ";
            std::getline(std::cin, valor);

            Receta* r = recetario.buscarRecetaPorNombre(valor);
            if (!r) std::cout << "No encontrada.\n";
            else std::cout << r->toString() << "\n";
        }
        else if (opcion == "categoria") {
```

```
            std::cout << "Categoria: ";
            std::getline(std::cin, valor);

            recetario.buscarPorCategoria(valor);

        } else {
            std::cout << "Opcion invalida.\n";
        }

    } catch (...) {
        std::cout << "Error en busqueda.\n";
    }
}


void Menu::eliminarReceta() {
    std::string nombre;
    std::cin.ignore();
    std::cout << "Nombre a eliminar: ";
    std::getline(std::cin, nombre);

    try {
        recetario.eliminarRecetaPorNombre(nombre);
        std::cout << "Receta eliminada.\n";
    } catch (const std::exception& e) {
        std::cout << "Error: " << e.what() << "\n";
    }
}

void Menu::eliminarTodasRecetas() {
    recetario.eliminarTodasRecetas();
    std::cout << "Todas las recetas eliminadas.\n";
}

void Menu::ordenarRecetas() {
    std::string opcion;
    std::cin.ignore();
    std::cout << "Ordenar por (nombre/tiempo): ";
    std::getline(std::cin, opcion);

    if (opcion == "nombre") recetario.ordenarPorNombre();
    else if (opcion == "tiempo") recetario.ordenarPorTiempo();
    else std::cout << "Opcion invalida.\n";

    mostrarRecetas();
```

```
}

void Menu::editarReceta() {
    try {
        std::string nombre;
        std::cin.ignore();
        std::cout << "Nombre a editar: ";
        std::getline(std::cin, nombre);

        Receta* r = recetario.buscarRecetaPorNombre(nombre);
        if (!r) {
            std::cout << "No encontrada.\n";
            return;
        }

        int opc;
        do {
            std::cout <<
"\n1.Nombre\n2.Categoria\n3.Autor\n4.Tiempo\n5.Procedimiento\n"
                         "6.Reemplazar ingredientes\n0.Salir\n";

            std::cout << "Opcion: ";
            std::cin >> opc;
            std::cin.ignore();

            if (opc == 1) {
                std::string x; std::getline(std::cin, x); r->setNombre(x);
            }
            else if (opc == 2) {
                std::string x; std::getline(std::cin, x); r->setCategoria(x);
            }
            else if (opc == 3) {
                std::string x; std::getline(std::cin, x); r->setAutor(x);
            }
            else if (opc == 4) {
                int t; std::cin >> t; r->setTiempo(t); std::cin.ignore();
            }
            else if (opc == 5) {
                std::string x; std::getline(std::cin, x); r->setProcedimiento(x);
            }
            else if (opc == 6) {
                r->eliminarTodosIngredientes();
                int n;
                std::cout << "Num ingredientes: ";
                std::cin >> n;
                std::cin.ignore();
```

```
                for (int i = 0; i < n; i++) {
                    std::string n2, u;
                    float c;

                    std::cout << "Nombre: ";
                    std::getline(std::cin, n2);

                    std::cout << "Unidad: ";
                    std::getline(std::cin, u);

                    std::cout << "Cantidad: ";
                    std::cin >> c;
                    std::cin.ignore();

                    r->agregarIngrediente(Ingrediente(n2, u, c));
                }
            }

        } while (opc != 0);

    } catch (...) {
        std::cout << "Error editando.\n";
    }
}

void Menu::guardarRecetario() {
    std::string archivo;
    std::cin.ignore();
    std::cout << "Archivo: ";
    std::getline(std::cin, archivo);

    try {
        recetario.guardarADisco(archivo);
        std::cout << "Guardado.\n";
    } catch (const std::exception& e) {
        std::cout << "Error: " << e.what() << "\n";
    }
}

void Menu::leerRecetario() {
    std::string archivo;
    std::cin.ignore();
    std::cout << "Archivo: ";
    std::getline(std::cin, archivo);
```

```
    try {
        recetario.leerDeDisco(archivo);
        std::cout << "Cargado.\n";
    } catch (const std::exception& e) {
        std::cout << "Error: " << e.what() << "\n";
    }
}

void Menu::iniciar() {
    int opc;
    do {
        std::cout << "\n--- MENU ---\n"
                  << "1.Agregar\n2.Mostrar\n3.Buscar\n4.Eliminar\n"
                  << "5.Eliminar todas\n6.Ordenar\n7.Editar\n"
                  << "8.Guardar\n9.Cargar\n0.Salir\n";
        std::cout << "Opcion: ";
        std::cin >> opc;

        switch (opc) {
            case 1: agregarReceta(); break;
            case 2: mostrarRecetas(); break;
            case 3: buscarReceta(); break;
            case 4: eliminarReceta(); break;
            case 5: eliminarTodasRecetas(); break;
            case 6: ordenarRecetas(); break;
            case 7: editarReceta(); break;
            case 8: guardarRecetario(); break;
            case 9: leerRecetario(); break;
        }

    } while (opc != 0);
}
```

4. ListaSimplementeLigada.hpp

```
#ifndef LISTA_SIMPLEMENTE_LIGADA_HPP
#define LISTA_SIMPLEMENTE_LIGADA_HPP

#include <iostream>
#include <sstream>
#include <stdexcept>

template <typename T>
class Nodo {
```

```
public:
    T dato;
    Nodo* next;

    Nodo(const T& d) : dato(d), next(nullptr) {}
};

template <typename T>
class ListaSimplementeLigada {
public:
    Nodo<T>* ancla;
    Nodo<T>* localiza;
    Nodo<T>* ultimo;

    ListaSimplementeLigada() {
        ancla = new Nodo<T>(T());
        ancla->next = nullptr;
        localiza = nullptr;
        ultimo = nullptr;
    }

    ListaSimplementeLigada(const ListaSimplementeLigada<T>& other) {
        ancla = new Nodo<T>(T());
        localiza = nullptr;
        ultimo = nullptr;

        Nodo<T>* p = other.ancla->next;
        Nodo<T>* prev = ancla;

        while (p != nullptr) {
            Nodo<T>* nuevo = new Nodo<T>(p->dato);
            prev->next = nuevo;
            prev = nuevo;
            ultimo = nuevo;
            p = p->next;
        }
    }

    ListaSimplementeLigada(ListaSimplementeLigada<T>&& other) noexcept {
        ancla = other.ancla;
        localiza = other.localiza;
        ultimo = other.ultimo;

        other.ancla = new Nodo<T>(T());
        other.localiza = nullptr;
        other.ultimo = nullptr;
```

```
    }

    ListaSimplementeLigada<T>& operator=(const ListaSimplementeLigada<T>& other) {
        if (this == &other) return *this;

        anula();

        Nodo<T>* p = other.ancla->next;
        Nodo<T>* prev = ancla;

        while (p != nullptr) {
            Nodo<T>* nuevo = new Nodo<T>(p->dato);
            prev->next = nuevo;
            prev = nuevo;
            ultimo = nuevo;
            p = p->next;
        }

        return *this;
    }

    ListaSimplementeLigada<T>& operator=(ListaSimplementeLigada<T>&& other)
noexcept {
        if (this == &other) return *this;

        ancla = other.ancla;
        localiza = other.localiza;
        ultimo = other.ultimo;

        other.ancla = new Nodo<T>(T());
        other.localiza = nullptr;
        other.ultimo = nullptr;

        return *this;
    }

    ~ListaSimplementeLigada() {
        anula();
        delete ancla;
    }

    bool vacia() const {
        return ancla->next == nullptr;
    }

    Nodo<T>* first() const {
```

```
    if (vacia()) return nullptr;
    return ancla->next;
}


void insertarOrdenado(const T& dato) {
    Nodo<T>* nuevo = new Nodo<T>(dato);
    Nodo<T>* ant = ancla;
    Nodo<T>* act = ancla->next;

    while (act && act->dato < dato) {
        ant = act;
        act = act->next;
    }

    ant->next = nuevo;
    nuevo->next = act;

    if (act == nullptr)
        ultimo = nuevo;
}

void elimina(const T& valor) {
    if (vacia()) throw std::runtime_error("La lista está vacía.");
    Nodo<T>* ant = ancla;
    Nodo<T>* act = ancla->next;
    while (act && !(act->dato == valor)) {
        ant = act;
        act = act->next;
    }
    if (!act) throw std::runtime_error("Elemento no encontrado.");
    if (act == ultimo) {
        ultimo = (ant == ancla ? nullptr : ant);
    }
    ant->next = act->next;
    delete act;
}

Nodo<T>* localizar(const T& dato) const {
    Nodo<T>* p = ancla->next;
    while (p && !(p->dato == dato)) p = p->next;
    return p;
}

void anula() {
    Nodo<T>* p = ancla->next;
```

```
        while (p != nullptr) {
            Nodo<T>* aux = p;
            p = p->next;
            delete aux;
        }

        ancla->next = nullptr;
        ultimo = nullptr;
        localiza = nullptr;
    }

    std::string toString() const {
        std::ostringstream oss;
        Nodo<T>* p = ancla->next;
        int i = 1;

        while (p != nullptr) {
            oss << i++ << ". " << p->dato << "\n";
            p = p->next;
        }

        return oss.str();
    }
};

#endif
```

5. ListaDoblementeLigada.hpp

```
#ifndef LISTA_DOBLEMENTE_LIGADA_HPP
#define LISTA_DOBLEMENTE_LIGADA_HPP

#include <iostream>
#include <sstream>
#include <stdexcept>

template <typename T>
class NodoDoble {
public:
    T dato;
    NodoDoble<T>* siguiente;
    NodoDoble<T>* anterior;
    NodoDoble(const T& d) : dato(d), siguiente(nullptr), anterior(nullptr) {}
};

template <typename T>
class ListaDoblementeLigada {
public:
    NodoDoble<T>* ancla;
    NodoDoble<T>* localiza;
    NodoDoble<T>* ultimo;

    ListaDoblementeLigada() {
        ancla = new NodoDoble<T>(T());
        ancla->siguiente = nullptr;
        ancla->anterior = nullptr;
        localiza = nullptr;
        ultimo = nullptr;
    }

    ~ListaDoblementeLigada() {
        anula();
        delete ancla;
    }

    bool vacia() const {
        return ancla->siguiente == nullptr;
    }

    NodoDoble<T>* first() const {
        if (vacia()) throw std::runtime_error("La lista esta vacia");
```

```
        return ancla->siguiente;
    }

    NodoDoble<T>* last() const {
        if (vacia()) throw std::runtime_error("La lista esta vacia");
        return ultimo;
    }

    NodoDoble<T>* nextNode(NodoDoble<T>* nodo) const {
        if (!nodo) throw std::runtime_error("Nodo invalido");
        return nodo->siguiente;
    }

    NodoDoble<T>* previousNode(NodoDoble<T>* nodo) const {
        if (!nodo) throw std::runtime_error("Nodo invalido");
        if (nodo == ancla->siguiente) return ancla;
        if (!nodo->anterior) throw std::runtime_error("Nodo no pertenece a la
lista");
        return nodo->anterior;
    }

    NodoDoble<T>* localizaValor(const T& valor) {
        NodoDoble<T>* p = ancla->siguiente;
        while (p != nullptr) {
            if (p->dato == valor) {
                localiza = p;
                return p;
            }
            p = p->siguiente;
        }
        localiza = nullptr;
        return nullptr;
    }

void insertar(const T& dato) {
    if (!localiza) localiza = ancla;
    NodoDoble<T>* nuevo = new NodoDoble<T>(dato);
    NodoDoble<T>* siguiente = localiza->siguiente;
    nuevo->siguiente = siguiente;
    nuevo->anterior = localiza;
    localiza->siguiente = nuevo;
    if (siguiente) siguiente->anterior = nuevo;
    if (localiza == ultimo) ultimo = nuevo;
    localiza = nuevo;
}
void insertarEnPosicion(const T& dato, int pos) {
```

```
    NodoDoble<T>* p = ancla;
    int contador = 0;
    while (p->siguiente && contador < pos) {
        p = p->siguiente;
        contador++;
    }
    localiza = p;
    insertar(dato);
}
void insertarAlFinal(const T& dato) {
    NodoDoble<T>* nuevo = new NodoDoble<T>(dato);
    if (!ancla->siguiente) {
        ancla->siguiente = nuevo;
        nuevo->anterior = ancla;
        ultimo = nuevo;
    } else {
        ultimo->siguiente = nuevo;
        nuevo->anterior = ultimo;
        ultimo = nuevo;
    }
}

    void elimina(const T& valor) {
        if (vacia()) throw std::runtime_error("La lista esta vacia");
        NodoDoble<T>* p = ancla->siguiente;
        while (p != nullptr && !(p->dato == valor)) p = p->siguiente;
        if (!p) throw std::runtime_error("Elemento no encontrado");
        NodoDoble<T>* A = p->anterior;
        NodoDoble<T>* S = p->siguiente;
        A->siguiente = S;
        if (S) S->anterior = A;
        if (p == ultimo) {
            if (A == ancla) ultimo = nullptr;
            else ultimo = A;
        }
        delete p;
        localiza = nullptr;
    }

    NodoDoble<T>* dividirMitad(NodoDoble<T>* cabeza) {
        if (!cabeza || !cabeza->siguiente) return nullptr;
        NodoDoble<T>* lento = cabeza;
        NodoDoble<T>* rapido = cabeza->siguiente;
        while (rapido && rapido->siguiente) {
            lento = lento->siguiente;
            rapido = rapido->siguiente->siguiente;
```

```
    }
    NodoDoble<T>* mitad = lento->siguiente;
    lento->siguiente = nullptr;
    if (mitad) mitad->anterior = nullptr;
    return mitad;
}

NodoDoble<T>* mergeNombre(NodoDoble<T>* a, NodoDoble<T>* b) {
    if (!a) return b;
    if (!b) return a;
    if (a->dato.getNombre() < b->dato.getNombre()) {
        a->siguiente = mergeNombre(a->siguiente, b);
        if (a->siguiente) a->siguiente->anterior = a;
        a->anterior = nullptr;
        return a;
    } else {
        b->siguiente = mergeNombre(a, b->siguiente);
        if (b->siguiente) b->siguiente->anterior = b;
        b->anterior = nullptr;
        return b;
    }
}

NodoDoble<T>* mergeSortNombre(NodoDoble<T>* cabeza) {
    if (!cabeza || !cabeza->siguiente) return cabeza;
    NodoDoble<T>* mitad = dividirMitad(cabeza);
    NodoDoble<T>* izq = mergeSortNombre(cabeza);
    NodoDoble<T>* der = mergeSortNombre(mitad);
    return mergeNombre(izq, der);
}

NodoDoble<T>* mergeTiempo(NodoDoble<T>* a, NodoDoble<T>* b) {
    if (!a) return b;
    if (!b) return a;
    if (a->dato.getTiempo() < b->dato.getTiempo()) {
        a->siguiente = mergeTiempo(a->siguiente, b);
        if (a->siguiente) a->siguiente->anterior = a;
        a->anterior = nullptr;
        return a;
    } else {
        b->siguiente = mergeTiempo(a, b->siguiente);
        if (b->siguiente) b->siguiente->anterior = b;
        b->anterior = nullptr;
        return b;
    }
}
```

```
NodoDoble<T>* mergeSortTiempo(NodoDoble<T>* cabeza) {
    if (!cabeza || !cabeza->siguiente) return cabeza;
    NodoDoble<T>* mitad = dividirMitad(cabeza);
    NodoDoble<T>* izq = mergeSortTiempo(cabeza);
    NodoDoble<T>* der = mergeSortTiempo(mitad);
    return mergeTiempo(izq, der);
}

void anula() {
    NodoDoble<T>* p = ancla->siguiente;
    while (p != nullptr) {
        NodoDoble<T>* aux = p;
        p = p->siguiente;
        delete aux;
    }
    ancla->siguiente = nullptr;
    ultimo = nullptr;
    localiza = nullptr;
}

T recuperar(NodoDoble<T>* nodo) const {
    if (!nodo) throw std::runtime_error("Nodo invalido");
    return nodo->dato;
}

int size() const {
    int c = 0;
    NodoDoble<T>* p = ancla->siguiente;
    while (p != nullptr) { c++; p = p->siguiente; }
    return c;
}

std::string toString() const {
    std::ostringstream oss;
    NodoDoble<T>* p = ancla->siguiente;
    int i = 1;
    while (p != nullptr) {
        try { oss << i << ". " << p->dato.toString() << "\n"; }
        catch (...) { oss << i << ". [dato]\n"; }
        p = p->siguiente;
        i++;
    }
    return oss.str();
}
```

```
    ListaDoblementeLigada<T>& operator=(const ListaDoblementeLigada<T>& other) {
        if (this == &other) return *this;
        anula();
        NodoDoble<T>* p = other.ancla->siguiente;
        NodoDoble<T>* ult = nullptr;
        while (p != nullptr) {
            NodoDoble<T>* nuevo = new NodoDoble<T>(p->dato);
            if (!ult) {
                ancla->siguiente = nuevo;
                nuevo->anterior = ancla;
            } else {
                ult->siguiente = nuevo;
                nuevo->anterior = ult;
            }
            ult = nuevo;
            p = p->siguiente;
        }
        ultimo = ult;
        localiza = nullptr;
        return *this;
    }
};

#endif
```

6. Ingrediente.hpp

```
#ifndef INGREDIENTE_HPP
#define INGREDIENTE_HPP

#include <string>
#include <sstream>
#include <iomanip>

class Ingrediente {
private:
    std::string nombre;
    std::string unidad;
    float cantidad;

public:
    Ingrediente();
    Ingrediente(const std::string& n, const std::string& u, float c);

    std::string getNombre() const;
```

```
    std::string getUnidad() const;
    float getCantidad() const;

    void setNombre(const std::string& n);
    void setUnidad(const std::string& u);
    void setCantidad(float c);

    std::string toString() const;

    bool operator==(const Ingrediente& other) const;
    bool operator<(const Ingrediente& other) const;
};

#endif
```

7. Ingrediente.cpp

```
#include "Ingrediente.hpp"
#include <sstream>
#include <iomanip>

Ingrediente::Ingrediente() : nombre(""), unidad(""), cantidad(0.0f) {}

Ingrediente::Ingrediente(const std::string& n, const std::string& u, float c)
    : nombre(n), unidad(u), cantidad(c) {}

std::string Ingrediente::getNombre() const { return nombre; }
std::string Ingrediente::getUnidad() const { return unidad; }
float Ingrediente::getCantidad() const { return cantidad; }

void Ingrediente::setNombre(const std::string& n) { nombre = n; }
void Ingrediente::setUnidad(const std::string& u) { unidad = u; }
void Ingrediente::setCantidad(float c) { cantidad = c; }

std::string Ingrediente::toString() const {
    std::ostringstream oss;
    oss << std::fixed << std::setprecision(2);
    oss << nombre << " - " << cantidad << " " << unidad;
    return oss.str();
}

bool Ingrediente::operator==(const Ingrediente& other) const {
    return nombre == other.nombre;
}

bool Ingrediente::operator<(const Ingrediente& other) const {
    return nombre < other.nombre;
```

```
}
```

8. Receta.hpp

```
#ifndef RECETA_HPP
#define RECETA_HPP

#include <string>
#include <iostream>
#include "ListaSimplementeLigada.hpp"
#include "Ingrediente.hpp"

class Receta {
private:
    std::string nombre;
    std::string categoria;
    std::string autor;
    int tiempo;
    std::string procedimiento;
    ListaSimplementeLigada<Ingrediente> ingredientes;

public:
    Receta();
    Receta(const std::string& n, const std::string& c,
           const std::string& a, int t, const std::string& p);
    Receta(const Receta& other);
    Receta(Receta&& other) noexcept;

    Receta& operator=(const Receta& other);
    Receta& operator=(Receta&& other) noexcept;
    bool operator==(const Receta& other) const;
    bool operator<(const Receta& other) const;

    const std::string& getNombre() const;
    const std::string& getCategoria() const;
    const std::string& getAutor() const;
    int getTiempo() const;
    const std::string& getProcedimiento() const;

    void setNombre(const std::string& n);
    void setCategoria(const std::string& c);
    void setAutor(const std::string& a);
    void setTiempo(int t);
    void setProcedimiento(const std::string& p);

    void agregarIngrediente(const Ingrediente& ing);
    void eliminarIngrediente(const std::string& nombreIng);
```

```
    void eliminarTodosIngredientes();
    Ingrediente* buscarIngrediente(const std::string& nombreIng);
    void imprimirIngredientes() const;

    std::string toString() const;
    std::string toStringResumen() const;

    void serializar(std::ostream& os) const;
    static Receta deserializar(std::istream& is);
};

#endif
```

9. Receta.cpp

```
#include "Receta.hpp"
#include <sstream>
#include <stdexcept>
#include <iostream>

Receta::Receta() : nombre(""), categoria(""), autor(""), tiempo(0),
procedimiento("") {}

Receta::Receta(const std::string& n, const std::string& c,
               const std::string& a, int t, const std::string& p)
    : nombre(n), categoria(c), autor(a), tiempo(t), procedimiento(p) {}

Receta::Receta(const Receta& other)
    : nombre(other.nombre), categoria(other.categoria), autor(other.autor),
      tiempo(other.tiempo), procedimiento(other.procedimiento) {
    Nodo<Ingrediente>* p = other.ingredientes.first();
    while (p) {
        ingredientes.insertarOrdenado(p->dato);
        p = p->next;
    }
}

Receta::Receta(Receta&& other) noexcept
    : nombre(std::move(other.nombre)), categoria(std::move(other.categoria)),
      autor(std::move(other.autor)), tiempo(other.tiempo),
      procedimiento(std::move(other.procedimiento)),
      ingredientes(std::move(other.ingredientes)) {}

Receta& Receta::operator=(const Receta& other) {
```

```
    if (this == &other) return *this;
    nombre = other.nombre;
    categoria = other.categoria;
    autor = other.autor;
    tiempo = other.tiempo;
    procedimiento = other.procedimiento;

    ingredientes.anula();
    Nodo<Ingrediente>* p = other.ingredientes.first();
    while (p) {
        ingredientes.insertarOrdenado(p->dato);
        p = p->next;
    }
    return *this;
}

Receta& Receta::operator=(Receta&& other) noexcept {
    if (this == &other) return *this;
    nombre = std::move(other.nombre);
    categoria = std::move(other.categoria);
    autor = std::move(other.autor);
    tiempo = other.tiempo;
    procedimiento = std::move(other.procedimiento);
    ingredientes = std::move(other.ingredientes);
    return *this;
}

bool Receta::operator==(const Receta& other) const { return nombre == other.nombre;
}
bool Receta::operator<(const Receta& other) const { return nombre < other.nombre; }

const std::string& Receta::getNombre() const { return nombre; }
const std::string& Receta::getCategoria() const { return categoria; }
const std::string& Receta::getAutor() const { return autor; }
int Receta::getTiempo() const { return tiempo; }
const std::string& Receta::getProcedimiento() const { return procedimiento; }

void Receta::setNombre(const std::string& n) { nombre = n; }
void Receta::setCategoria(const std::string& c) { categoria = c; }
void Receta::setAutor(const std::string& a) { autor = a; }
void Receta::setTiempo(int t) { tiempo = t; }
void Receta::setProcedimiento(const std::string& p) { procedimiento = p; }

void Receta::agregarIngrediente(const Ingrediente& ing) {
ingredientes.insertarOrdenado(ing); }
```

```
void Receta::eliminarIngrediente(const std::string& nombreIng) {
    Ingrediente temp(nombreIng, "", 0);
    Nodo<Ingrediente>* p = ingredientes.localizar(temp);
    if (!p) throw std::runtime_error("Ingrediente no encontrado.");
    ingredientes.elimina(p->dato);
}

void Receta::eliminarTodosIngredientes() { ingredientes.anula(); }

Ingrediente* Receta::buscarIngrediente(const std::string& nombreIng) {
    Ingrediente temp(nombreIng, "", 0);
    Nodo<Ingrediente>* p = ingredientes.localizar(temp);
    if (!p) return nullptr;
    return &p->dato;
}

void Receta::imprimirIngredientes() const {
    Nodo<Ingrediente>* p = ingredientes.first();
    while (p) {
        std::cout << "- " << p->dato.toString() << "\n";
        p = p->next;
    }
}

std::string Receta::toString() const {
    std::ostringstream oss;
    oss << "=== " << nombre << " ===\n";
    oss << "Categoría: " << categoria << "\n";
    oss << "Autor: " << autor << "\n";
    oss << "Tiempo: " << tiempo << " min\n\n";

    oss << "Ingredientes:\n";
    Nodo<Ingrediente>* p = ingredientes.first();
    while (p) {
        oss << "  - " << p->dato.toString() << "\n";
        p = p->next;
    }

    oss << "\nProcedimiento:\n";
    oss << "" << procedimiento << "\n";

    return oss.str();
}


std::string Receta::toStringResumen() const {
```

```
    std::ostringstream oss;
    oss << nombre << " [" << categoria << "] - " << tiempo << " min";
    return oss.str();
}

void Receta::serializar(std::ostream& os) const {
    os << nombre << "@" << categoria << "@" << autor << "@" << tiempo << "@" <<
procedimiento << "|";
    Nodo<Ingrediente>* p = ingredientes.first();
    while (p) {
        os << p->dato.getNombre() << "@" << p->dato.getUnidad() << "@" <<
p->dato.getCantidad();
        if (p->next) os << "&";
        p = p->next;
    }
    os << "%\n";
}
Receta Receta::deserializar(std::istream& is) {
    std::string linea;
    if (!std::getline(is, linea))
        throw std::runtime_error("Línea inválida.");

    size_t pos = linea.find('|');
    if (pos == std::string::npos)
        throw std::runtime_error("Formato de receta incorrecto.");

    std::string datosReceta = linea.substr(0, pos);
    std::string datosIng = linea.substr(pos + 1);

    std::stringstream ss(datosReceta);
    std::string n, c, a, tStr, proc;
    std::getline(ss, n, '@');
    std::getline(ss, c, '@');
    std::getline(ss, a, '@');
    std::getline(ss, tStr, '@');
    std::getline(ss, proc, '@');
    int t = std::stoi(tStr);

    Receta r(n, c, a, t, proc);

    if (!datosIng.empty() && datosIng.back() == '%')
        datosIng.pop_back();

    std::stringstream si(datosIng);
    std::string ingStr;
    while (std::getline(si, ingStr, '&')) {
```

```
        if (!ingStr.empty() && ingStr[0] == ' ')
            ingStr.erase(0, 1);

        std::stringstream sx(ingStr);
        std::string in, un, cantStr;
        std::getline(sx, in, '@');
        std::getline(sx, un, '@');
        std::getline(sx, cantStr, '@');
        float cant = std::stof(cantStr);
        r.agregarIngrediente(Ingrediente(in, un, cant));
    }

    return r;
}
```

10. Recetario.hpp

```
#ifndef RECETARIO_HPP
#define RECETARIO_HPP

#include "Receta.hpp"
#include "ListaDoblementeLigada.hpp"
#include "ListaSimplementeLigada.hpp"
#include <string>
#include <iostream>

class Recetario {
private:
    ListaDoblementeLigada<Receta> recetas;

public:
    Recetario();
    ~Recetario();

    void agregarReceta(const Receta& r);
    void agregarRecetaEnPosicion(const Receta& r, int pos);
    void eliminarRecetaPorNombre(const std::string& nombre);
    void eliminarTodasRecetas();
    Receta* buscarRecetaPorNombre(const std::string& nombre) const;
    void buscarPorCategoria(const std::string& categoria);


    void mostrarTodas() const;
    void ordenarPorNombre();
    void ordenarPorTiempo();
```

```
    void guardarADisco(const std::string& archivo) const;
    void leerDeDisco(const std::string& archivo);
};

#endif
```

11. Recetario.cpp

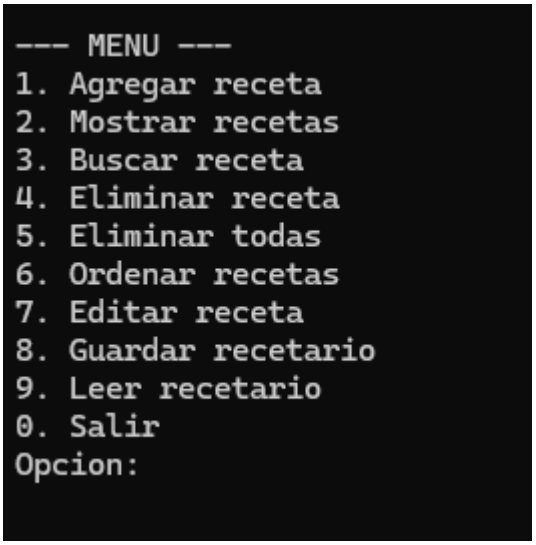
```
#include "Recetario.hpp"
#include <fstream>
#include <sstream>
#include <stdexcept>

Recetario::Recetario() {}
Recetario::~Recetario() { recetas.anula(); }

void Recetario::agregarReceta(const Receta& r) {
    recetas.localiza = recetas.ultimo;
    recetas.insertar(r);
}

void Recetario::agregarRecetaEnPosicion(const Receta& r, int pos) {
     NodoDoble<Receta>* p = recetas.ancla->siguiente;
    int contador = 1;
    while (p && contador < pos) {
        p = p->siguiente;
        contador++;
    }

    if (!p) p = recetas.ultimo;

    recetas.localiza = p->anterior ? p->anterior : recetas.ancla;
    recetas.insertarEnPosicion(r, pos);
}



void Recetario::eliminarRecetaPorNombre(const std::string& nombre) {
    NodoDoble<Receta>* p = recetas.ancla->siguiente;
    while (p) {
        if (p->dato.getNombre() == nombre) {
            recetas.elimina(p->dato);
            return;
        }
        p = p->siguiente;
    }
```

```
    throw std::runtime_error("Receta no encontrada");
}

void Recetario::eliminarTodasRecetas() {
    recetas.anula();
}

Receta* Recetario::buscarRecetaPorNombre(const std::string& nombre) const {
    NodoDoble<Receta>* p = recetas.ancla->siguiente;
    while (p) {
        if (p->dato.getNombre() == nombre) return &p->dato;
        p = p->siguiente;
    }
    return nullptr;
}

void Recetario::buscarPorCategoria(const std::string& categoria) {
    NodoDoble<Receta>* p = recetas.first();
    bool encontrado = false;

    while (p) {
        Receta& r = p->dato;

        if (r.getCategoria() == categoria) {
            std::cout << r.toString() << "\n";
            std::cout << "-----------------------------------------------\n";
            encontrado = true;
        }

        p = p->siguiente;
    }

    if (!encontrado) {
        std::cout << "No se encontraron recetas en esa categoria.\n";
    }
}




void Recetario::mostrarTodas() {
    if (recetas.vacia()) {
        std::cout << "No hay recetas.\n";
        return;
    }
```

```
    std::cout << "No.  "
              << std::left << std::setw(30) << "Nombre"
              << std::setw(30) << "Categoria"
              << std::setw(30) << "Autor"
              << std::setw(15) << "Tiempo"
              << "\n";

    std::cout <<
"-------------------------------------------------------------------------------------
----------------\n";

    NodoDoble<Receta>* p = recetas.first();
    int num = 1;

    while (p != nullptr) {
        Receta r = p->dato;

        std::cout << std::setw(4) << num++
                  << std::left << std::setw(30) << r.getNombre()
                  << std::setw(30) << r.getCategoria()
                  << std::setw(30) << r.getAutor()
                  << std::setw(15) << r.getTiempo()
                  << "\n";

        p = p->siguiente;
    }
}


void Recetario::ordenarPorNombre() {
    if (recetas.vacia()) return;
    NodoDoble<Receta>* nueva = recetas.mergeSortNombre(recetas.ancla->siguiente);
    recetas.ancla->siguiente = nueva;
    NodoDoble<Receta>* p = nueva;
    NodoDoble<Receta>* ant = recetas.ancla;
    while (p) {
        p->anterior = ant;
        ant = p;
        p = p->siguiente;
    }
    recetas.ultimo = ant;
}

void Recetario::ordenarPorTiempo() {
    if (recetas.vacia()) return;
    NodoDoble<Receta>* nueva = recetas.mergeSortTiempo(recetas.ancla->siguiente);
```

```
    recetas.ancla->siguiente = nueva;
    NodoDoble<Receta>* p = nueva;
    NodoDoble<Receta>* ant = recetas.ancla;
    while (p) {
        p->anterior = ant;
        ant = p;
        p = p->siguiente;
    }
    recetas.ultimo = ant;
}

void Recetario::guardarADisco(const std::string& archivo) const {
    std::ofstream os(archivo);
    if (!os) throw std::runtime_error("No se pudo abrir archivo para escritura");
    NodoDoble<Receta>* p = recetas.ancla->siguiente;
    while (p) {
        p->dato.serializar(os);
        p = p->siguiente;
    }
}
void Recetario::leerDeDisco(const std::string& archivo) {
    std::ifstream is(archivo);
    if (!is) throw std::runtime_error("No se pudo abrir archivo para lectura");

    std::string linea;
    while (std::getline(is, linea)) {
        if (linea.size() < 3) continue;
        std::stringstream ss(linea);
        Receta r = Receta::deserializar(ss);

        recetas.localiza = recetas.ultimo;
        recetas.insertarAlFinal(r);
    }
}
```

4. Pruebas



1. Menú principal del proyecto.

```
Opcion: 1
Nombre: Lasaña Tradicional
Categoria (Desayuno/Comida/Cena/Navide▒o): Comida
Autor: Chef Carlos
Tiempo de preparacion (min): 75
Procedimiento: Preparar la salsa: Sofríe la cebolla picada y el ajo en el aceite de oliva ha
sta que estén dorados. Agrega la carne molida y cocina hasta que cambie de color. Después in
corpora la salsa de tomate, sal y pimienta. Cocina a fuego medio durante 15 minutos. Prepara
r la pasta: Cocina las láminas de lasaña en agua con sal hasta que estén al dente. Escúrrela
s y resérvalas. Armar la lasaña: En un molde refractario coloca una capa de pasta, luego car
ne con salsa, un poco de bechamel y queso mozzarella. Repite las capas hasta llenar el molde
. Termina con una capa de queso mozzarella y parmesano por encima. Hornear: Lleva al horno p
recalentado a 180 °C durante 25–30 minutos, hasta que el queso esté dorado y burbujeante. De
ja reposar 10 minutos antes de servir para que mantenga su forma.
Numero de ingredientes: 10
Ingrediente 1 nombre: Pasta para lasaña
Unidad: laminas
Cantidad: 12
Ingrediente 2 nombre: Carne molida de res
Unidad: g
Cantidad: 500
Ingrediente 3 nombre: Salsa de tomate
Unidad: ml
Cantidad: 400
Ingrediente 4 nombre: Queso mozzarella
Unidad: g
Cantidad: 250
Ingrediente 5 nombre: Queso Parmesano
Unidad: g
Cantidad: 50
Ingrediente 6 nombre: Cebolla
Unidad: u
Cantidad: 1
Ingrediente 7 nombre: Ajo
Unidad: dientes
Cantidad: 2
Ingrediente 8 nombre: Aceite de oliva
Unidad: ml
Cantidad: 30
Ingrediente 9 nombre: Sal
Unidad: g
Cantidad: 5
Ingrediente 10 nombre: Pimienta negra
Unidad: g
Cantidad: 2.5
Posicion donde agregar (0 para final): 1
Receta agregada exitosamente.
```

2. Insertando receta (10 ingredientes).

```
--- MENU ---
1. Agregar receta
2. Mostrar recetas
3. Buscar receta
4. Eliminar receta
5. Eliminar todas
6. Ordenar recetas
7. Editar receta
8. Guardar recetario
9. Leer recetario
0. Salir
Opcion: 9
Nombre de archivo para leer: Recetas.txt
Recetario cargado.
```

3. Cargamos Recetas del disco.

```
Opcion: 2
No.  Nombre                        Categoria                 Autor                     Tiempo
-------------------------------------------------------------------------------------------------
1    Lasanha Tradicional           Comida                    Chef Carlos               75
2    Tortilla Espanola             Desayuno                  Chef Maria                25
3    Ensalada Cesar                Comida                    Chef Luis                 15
4    Brownie Chocolate             Postre                    Chef Ana                  45
5    Sopa Pollo                    Comida                    Chef Juan                 60
6    Panques Vainilla              Desayuno                  Chef Laura                30
7    Smoothie Fresa                Bebida                    Chef Ana                  10
8    Pavo Horneado Tradicional     Navideno                  Chef Roberto              180
9    Hamburguesa Clasica           Comida                    Chef Luis                 35
10   Huevos Revueltos              Desayuno                  Chef Sofia                8
11   Sandwich de Jamon             Cena                      Chef Daniel               5
12   Omelette de Queso con Jamon   Desayuno                  Chef Maria                12
```

4. Mostrar lista de recetas.

```
Opcion: 3
Buscar por (nombre/categoria): nombre
Nombre: Lasanha Tradicional
=== Lasanha Tradicional ===
Categoria: Comida
Autor: Chef Carlos
Tiempo: 75 min

Ingredientes:
  - Aceite de oliva - 30.00 ml
  - Ajo - 2.00 dientes
  - Carne molida de res - 500.00 g
  - Cebolla - 1.00 u
  - Pasta para lasanha - 12.00 laminas
  - Pimienta negra - 2.50 g
  - Queso mozzarella - 250.00 g
  - Queso parmesano - 50.00 g
  - Sal - 5.00 g
  - Salsa de tomate - 400.00 ml

Procedimiento:
Preparar la salsa: Sofreir cebolla y ajo en aceite hasta dorar. Agregar carne molida, salsa de tomate, sal y pimienta. Cocinar 15 minutos. Cocinar laminas de lasanha
en agua con sal. Armar capas de pasta, carne, bechamel y queso. Hornear 25-30 minutos a 180 C.
```

6. Buscamos una receta por nombre (ingredientes ordenados por nombre).

```
Opcion: 3
Buscar por (nombre/categoria): categoria
Categoria: Desayuno
=== Tortilla Espanola ===
Categoria: Desayuno
Autor: Chef Maria
Tiempo: 25 min

Ingredientes:
  - Aceite de oliva - 50.00 ml
  - Aceite vegetal - 10.00 ml
  - Ajo - 1.00 dientes
  - Cebolla - 1.00 u
  - Huevo - 4.00 u
  - Leche - 15.00 ml
  - Papa - 3.00 u
  - Perejil - 2.00 g
  - Pimienta - 1.00 g
  - Sal - 2.00 g

Procedimiento:
Pelar y cortar papas. Freir hasta dorar. Mezclar con huevo y cebolla. Cocinar en sarten y dorar por ambos lados.

------------------------------------------------
=== Panques Vainilla ===
Categoria: Desayuno
Autor: Chef Laura
Tiempo: 30 min

Ingredientes:
  - Aceite vegetal - 20.00 ml
  - Azucar - 100.00 g
  - Esencia de vainilla - 5.00 ml
  - Harina - 200.00 g
  - Huevo - 2.00 u
  - Leche - 100.00 ml
  - Mantequilla - 20.00 g
  - Miel - 10.00 g
  - Polvo para hornear - 5.00 g
  - Sal - 1.00 g

Procedimiento:
Mezclar harina, azucar, huevos, leche y esencia de vainilla. Hornear 20 minutos a 180 C.

------------------------------------------------
=== Huevos Revueltos ===
Categoria: Desayuno
Autor: Chef Sofia
Tiempo: 8 min

Ingredientes:
  - Aceite vegetal - 5.00 ml
  - Ajo - 1.00 dientes
  - Huevo - 3.00 u
  - Leche - 20.00 ml
  - Mantequilla - 15.00 g
```

```
  - Perejil - 2.00 g
  - Pimienta - 1.00 g
  - Queso - 20.00 g
  - Sal - 1.00 g
  - Tomate - 1.00 u

Procedimiento:
Batir huevos con sal. Cocinar en sarten con mantequilla moviendo hasta que queden suaves.

------------------------------------------------
=== Omelette de Queso con Jamon ===
Categoria: Desayuno
Autor: Chef Maria
Tiempo: 12 min

Ingredientes:
  - Aceite vegetal - 5.00 ml
  - Huevo - 3.00 u
  - Jamon - 40.00 g
  - Leche - 15.00 ml
  - Mantequilla - 10.00 g
  - Perejil - 2.00 g
  - Pimienta - 1.00 g
  - Queso - 40.00 g
  - Sal - 1.00 g
  - Tomate - 1.00 u

Procedimiento:
Batir huevos con sal y pimienta. Cocinar en sarten con mantequilla, agregar jamon y queso, doblar el omelette y servir.

------------------------------------------------
```

7. Buscamos por categoría “Desayuno” y las ponemos todas en la pantalla.

```
--- MENU ---
1.Agregar
2.Mostrar
3.Buscar
4.Eliminar
5.Eliminar todas
6.Ordenar
7.Editar
8.Guardar
9.Cargar
0.Salir
Opcion: 4
Nombre a eliminar: Sandwich de Jamon
Receta eliminada.
```

8. Eliminamos una receta.

```
--- MENU ---
1.Agregar
2.Mostrar
3.Buscar
4.Eliminar
5.Eliminar todas
6.Ordenar
7.Editar
8.Guardar
9.Cargar
0.Salir
Opcion: 6
Ordenar por (nombre/tiempo): nombre
```

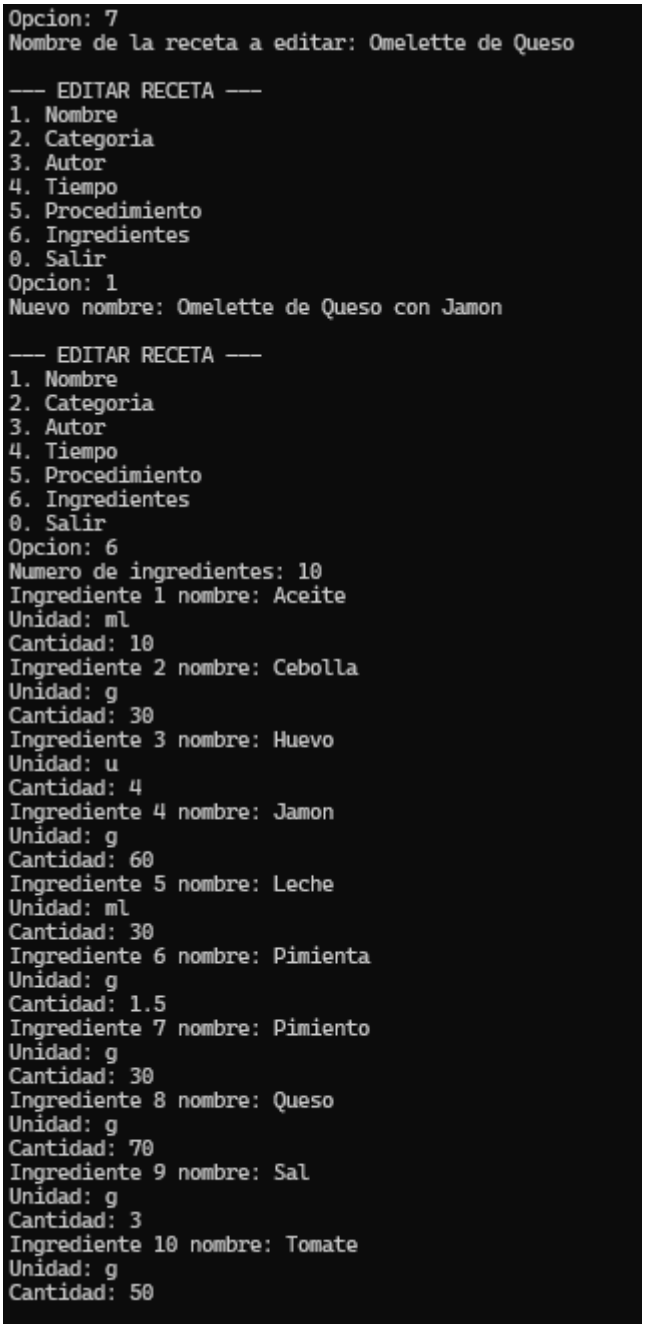
No.	Nombre	Categoria	Autor	Tiempo
1	Brownie Chocolate	Postre	Chef Ana	45
2	Ensalada Cesar	Comida	Chef Luis	15
3	Hamburguesa Clasica	Comida	Chef Luis	35
4	Huevos Revueltos	Desayuno	Chef Sofia	8
5	Lasanha Tradicional	Comida	Chef Carlos	75
6	Omelette de Queso con Jamon	Desayuno	Chef Maria	12
7	Panques Vainilla	Desayuno	Chef Laura	30
8	Pavo Horneado Tradicional	Navideno	Chef Roberto	180
9	Smoothie Fresa	Bebida	Chef Ana	10
10	Sopa Pollo	Comida	Chef Juan	60
11	Tortilla Espanola	Desayuno	Chef Maria	25

9. Ordenamiento por nombre.

```
Ordenar por (nombre/tiempo): tiempo
```

No.	Nombre	Categoria	Autor	Tiempo
1	Huevos Revueltos	Desayuno	Chef Sofia	8
2	Smoothie Fresa	Bebida	Chef Ana	10
3	Omelette de Queso con Jamon	Desayuno	Chef Maria	12
4	Ensalada Cesar	Comida	Chef Luis	15
5	Tortilla Espanola	Desayuno	Chef Maria	25
6	Panques Vainilla	Desayuno	Chef Laura	30
7	Hamburguesa Clasica	Comida	Chef Luis	35
8	Brownie Chocolate	Postre	Chef Ana	45
9	Sopa Pollo	Comida	Chef Juan	60
10	Lasanha Tradicional	Comida	Chef Carlos	75
11	Pavo Horneado Tradicional	Navideno	Chef Roberto	180

10. Ordenamos por tiempo.



11. Edición de algunos ingredientes de una receta.

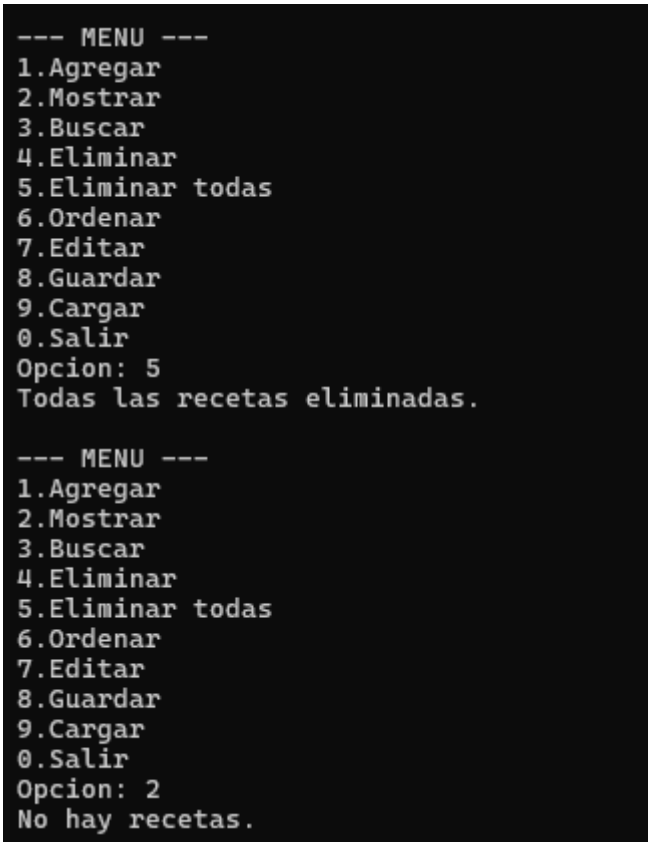
```
1.Nombre
2.Categoria
3.Autor
4.Tiempo
5.Procedimiento
6.Reemplazar ingredientes
0.Salir
Opcion: 1
Omelette Queso, Huevo y Jamon

1.Nombre
2.Categoria
3.Autor
4.Tiempo
5.Procedimiento
6.Reemplazar ingredientes
0.Salir
Opcion: 2
Desayuno

1.Nombre
2.Categoria
3.Autor
4.Tiempo
5.Procedimiento
6.Reemplazar ingredientes
0.Salir
Opcion: 3
Chef Sofia

1.Nombre
2.Categoria
3.Autor
4.Tiempo
5.Procedimiento
6.Reemplazar ingredientes
0.Salir
Opcion: 4
15
```

```
Opcion: 5
Batir huevos con sal, pimienta y un poco de leche. Derreti
r mantequilla en un sarten y verter la mezcla. Agregar que
so y jamon picado. Cocinar a fuego medio hasta que empiece
 a cuajar. Doblar el omelette y cocinar unos segundos mas.
 Servir caliente.
```

12. Edición de los demás atributos de la receta.

```
--- MENU ---
1.Agregar
2.Mostrar
3.Buscar
4.Eliminar
5.Eliminar todas
6.Ordenar
7.Editar
8.Guardar
9.Cargar
0.Salir
Opcion: 8
Archivo: Recetas.txt
Guardado.
```

12. Guardamos recetario al disco duro.



13. Eliminamos todas las recetas de la memoria.

5. Manual de Instalación

El presente manual describe los pasos necesarios para realizar la instalación correcta del sistema de Recetario en un entorno local. Incluye los requisitos indispensables, las configuraciones previas y el procedimiento detallado para compilar y ejecutar el programa. Este documento permite que cualquier usuario pueda instalar el software sin requerir conocimientos avanzados de programación.

1. Requisitos del sistema

Hardware

- Procesador: Intel/AMD de 1 GHz o superior.
- Memoria RAM: 512 MB mínimo, 1 GB recomendado.
- Almacenamiento disponible: 20–30 MB (suficiente para el ejecutable y recetas).
- Monitor con resolución mínima de 1280×720 (para poder visualizar mejor el texto).

Software

- Sistema operativo Windows 7, 8, 10 u 11 pues fue desarrollado en este entorno.
- Se recomienda tener permisos de usuario estándar para escribir archivos dentro de la carpeta del sistema.
- Instalar compilador Msys2
- Sin dependencias externas: No requiere frameworks adicionales ni instalaciones extra.
- Se necesita g++ 7 o superior (por el uso de C++17).
- Tener make o equivalente (para compilar todo de una vez).

2. Contenido del paquete entregado

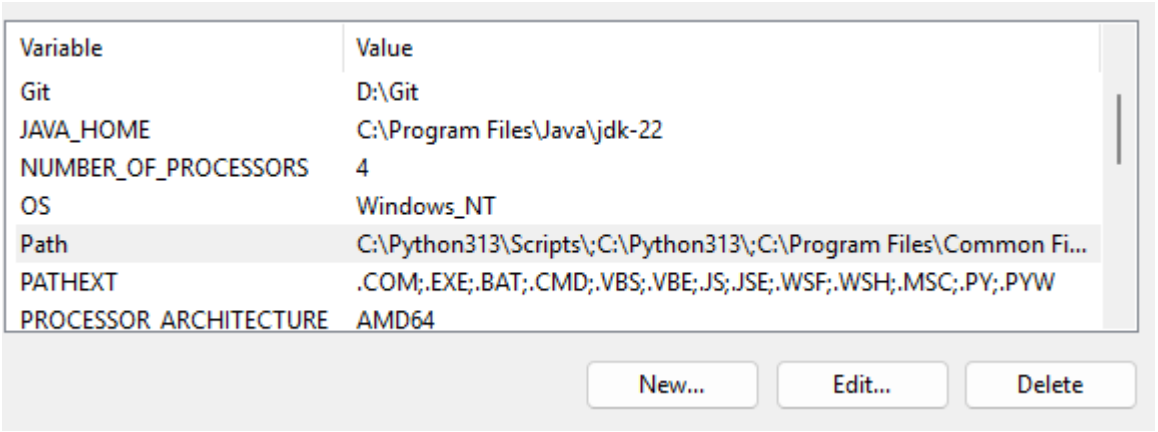
Al descargar el proyecto, encontrarás esta estructura:

RecetarioChef/

├─ .vscode/	**Archivos de configuración .json**
│	
├─ src/	**Códigos fuente (.cpp)**
├─ include/	**Cabeceras (.hpp)**
├─ lib/	**Librerías externas (no las hay)**
├─ output/	**Ejecutable compilado (se crea al compilar en esta carpeta)**
├─ Makefile	**Archivo para compilar y ejecutar**
└─ Recetas.txt	**Archivo de recetas de ejemplo para cargar y guardar**

3. Procedimiento de instalación

1. Copiar la carpeta completa RecetarioChef
2. Abrir VSCode desde la carpeta raíz (RecetarioChef)
3. Configurar PATH (solo Windows, si no lo tienen)
    - Abrir Panel de Control → Sistema → Configuración avanzada del sistema → Variables de entorno → PATH
    - Agregar rutas de MSYS2/MinGW:



6. Manual de Uso

1. Si no usaremos vscode para compilar el proyecto abrimos la carpeta del proyecto en la línea de comando.

```
C:\Users\marti>cd C:\Universidad\Codigos4to\Estructuras\ProyectoFinal\RecetarioChef
```

2. Lo compilamos usando el comando mingw32-make

```
C:\Universidad\Codigos4to\Estructuras\ProyectoFinal\RecetarioChef>mingw32-make
g++ -std=c++17 -Wall -Wextra -g -Iinclude -c -MMD src/Ingrediente.cpp -o src/Ingrediente.o
g++ -std=c++17 -Wall -Wextra -g -Iinclude -c -MMD src/Menu.cpp -o src/Menu.o
g++ -std=c++17 -Wall -Wextra -g -Iinclude -c -MMD src/Receta.cpp -o src/Receta.o
g++ -std=c++17 -Wall -Wextra -g -Iinclude -c -MMD src/Recetario.cpp -o src/Recetario.o
g++ -std=c++17 -Wall -Wextra -g -Iinclude -c -MMD src/main.cpp -o src/main.o
g++ -std=c++17 -Wall -Wextra -g src/Ingrediente.o src/Menu.o src/Receta.o src/Recetario.o src/main.o -o output\main.exe
"Compilaci├│n completa."
```

3. con eso ya tendríamos la puesta en marcha completa solo faltaria ejecutar el archivo main para poder usar el sistema.
4. Al ejecutar el programa aparece un menú con opciones, por ejemplo:

```
C:\Universidad\Codigos4to\Estructuras\ProyectoFinal\RecetarioChef>.\output\main.exe

--- MENU ---
1.Agregar
2.Mostrar
3.Buscar
4.Eliminar
5.Eliminar todas
6.Ordenar
7.Editar
8.Guardar
9.Cargar
0.Salir
Opcion:
```

Fig 6.1 Cada número corresponde a la acción que realizará el programa.

5. Descripción de funciones

- **Agregar receta:** ingresar nombre, ingredientes y pasos
- **Editar receta:** modificar datos de una receta existente
- **Eliminar receta:** borrar una receta
- **Buscar receta:** buscar por nombre o ingrediente
- **Listar recetas:** mostrar todas las recetas guardadas

- **Salir:** termina el programa y guarda los cambios en `Recetas.txt`

6. Archivo de datos

- **Recetas.txt**: contiene las recetas guardadas para persistencia

- Cada cambio realizado en el programa se guarda automáticamente al salir

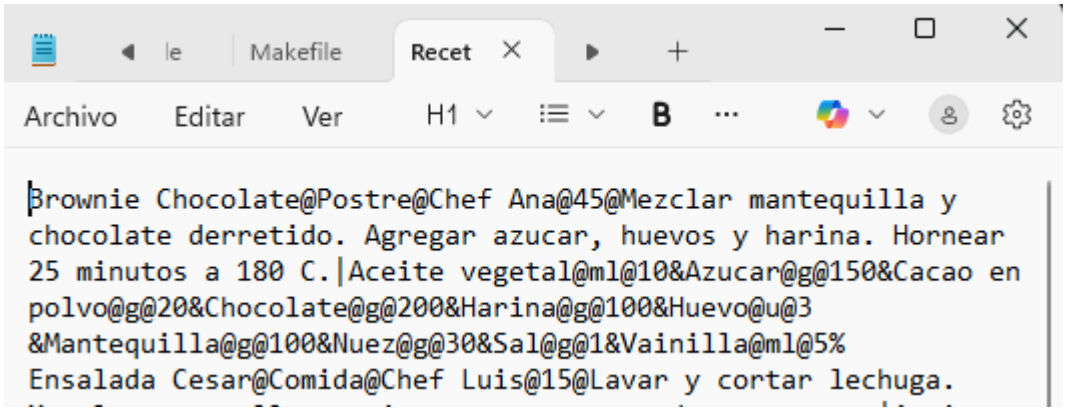


Fig 6.2 Este es un ejemplo de como se guarda una receta serializada.

7. Ejemplo de flujo de uso

    1. Ejecutar RecetarioChef

    2. Elegir opción 1 para agregar una receta

    3. Ingresar nombre: “Ensalada César”

    4. Ingresar ingredientes y pasos

    5. Elegir opción 5 para listar recetas y verificar que se agregó

    6. Salir con opción 6 → cambios guardados en Recetas.txt

7. Resumen Personal.

El desarrollo del recetario fue una manera en la que me permití aplicar todo lo que llevamos viendo en el semestre en un problema más enfocado a la vida real, me permitió enfrentar y superar varios retos técnicos complejos, al mismo tiempo que consolidaba mis conocimientos de manejo de datos. Uno de los principales desafíos fue la implementación de listas dinámicas para almacenar recetas e ingredientes de forma eficiente. Aprender a manejar la memoria dinámica correctamente fue una parte esencial, ya que debía asegurarme de que cada receta y cada ingrediente se almacenaran y liberaran adecuadamente, evitando fugas de memoria o errores de acceso.

Otro reto importante fue el manejo de archivos de texto con delimitadores, lo que implicaba desarrollar un sistema para serializar y deserializar datos de manera confiable. Cada receta debía poder cargarse y guardarse en Recetas.txt sin perder información ni corromper el formato, y esto requirió diseñar un método robusto para separar correctamente nombres, ingredientes y pasos usando delimitadores consistentes. Durante este proceso aprendí a trabajar con cadenas de texto, a procesar datos línea por línea y a validar que la información ingresada por el usuario cumpliera con los formatos esperados.

También enfrenté dificultades relacionadas con errores de compilación y compatibilidad del proyecto al probarlo en diferentes sistemas y con distintos compiladores. Al inicio, cambios de compiladores o la inclusión de objetos en carpetas distintas provocan errores que antes no existían y mas porque tuve que instalar de cero el compilador msys2, me obligó a entender a fondo cómo funcionan los Makefile, las rutas relativas. Aprendí a hacer el proyecto portable, asegurando que cualquier persona pudiera compilar y ejecutar el programa.

En conclusión, este proyecto me enseñó a integrar múltiples técnicas de escritura de codigo y documentación de este si lo podemos llamar así a este reporte de una manera práctica: listas dinámicas, memoria dinámica, manejo de archivos, serialización, delimitadores, validación de datos y compatibilidad entre sistemas. Fue un proyecto que no solo me permitió construir un programa funcional y robusto, sino que también me expuso a los retos reales que enfrenta un desarrollador al manejar datos complejos, depurar errores y garantizar la portabilidad del software y no solo enfocarse al código si no que hay muchos procesos que se deben realizar mucho antes incluso pensar en codificar como el diseño del programa donde controlamos el flujo, luego los uml para saber bien cada clase y ya comenzamos a codificar, tener esto muy claro hace que se sepa qué es lo que se quiere que realice el proyecto y tenemos la vista más clara, luego está el hacerle las pruebas correspondientes e incluso enseñarle a cualquier usuario a instalar y usar nuestro proyecto, estos conocimientos si fueron nuevos para mi y si aumentó muchísimo la idea que tenía yo de un proyecto de programación.