UNIVERSIDAD DE LAS AMÉRICAS

Facultad de ingeniería y ciencias aplicadas

Programación I

Tema

ProyectoFinal_Programacion_I

Tutor

Eddy Armas

Autores

Martin Aimacaña, Maikel Cachimuel

Periodo

2025

1. Caso de prueba

El usuario inicia el programa, carga 3 órdenes previas, registra una nueva validando que el código no exista y que el costo sea positivo, luego busca al cliente por su nombre, modifica sus horas de trabajo (lo cual actualiza el costo total automáticamente) y finalmente cierra el programa, lo que dispara el guardado automático en el CSV para la próxima sesión.

1.1 Nombre del sistema

Sistema de Gestión de Órdenes de Trabajo

1.2 Usuarios

Administrador de Taller: Encargado de supervisar todas las órdenes, verificar los costos totales generados y gestionar la persistencia de los datos en archivos.

Técnico de Soporte: Usuario que interactúa directamente con el sistema para registrar nuevos servicios, actualizar el progreso de las horas trabajadas y buscar información de clientes específicos.

1.3 Objetivos del Sistema

1.3.1 Objetivo General

Desarrollar una aplicación en lenguaje C que permita la gestión integral de órdenes de trabajo para servicios técnicos, garantizando que la información se mantenga guardada en archivos planos para futuras consultas.

1.3.2 Objetivos Específicos

- Gestión de Datos: Implementar las operaciones CRUD (Crear, Consultar, Actualizar y Eliminar) para administrar la información de los clientes y sus equipos.
- Cálculo Automatizado: Determinar el costo total del servicio aplicando una tarifa fija por hora ($10) sumada al costo base del trabajo realizado.
- Persistencia y Seguridad: Asegurar que los datos se guarden correctamente en un archivo de formato CSV con codificación UTF-8, permitiendo la carga automática al iniciar el programa.
- Validación de Integridad: Aplicar reglas de negocio estrictas, como la unicidad del código de orden y la restricción de valores positivos en los campos financieros.

1.4 Datos de entrada

- **Código orden:** Una cadena alfanumérica de entre 1 y 15 caracteres, sin espacios. Es el identificador único para cada registro. Mostrar el libro en la lista para verificar que fue agregado.
- **nombre cliente:** Cadena de texto que identifica al dueño del equipo.
- **Equipo:** Descripción del dispositivo.
- **Tipo de trabajo:** Descripción del servicio solicitado.
- **Costo base**: valor inicial o fijo asignado al servicio antes de sumar los cargos adicionales por tiempo trabajado.

- **horas trabajo**: variable que representa el tiempo invertido por el técnico en el equipo de computación.
- **Opciones del Menú**: A interfaz que permite al usuario interactuar con el sistema de forma organizada.

1.5 Datos de salida

Columna	Ejemplo de Valor	Origen del Dato
Código	SOP-2024	Ingreso de usuario.
Cliente	Carlos Mendoza	Ingreso de usuario.
Equipo	Servidor Dell	Ingreso de usuario.
Tipo Trabajo	Mantenimiento	Ingreso de usuario.
Costo Base	$50.00	Ingreso de usuario.
Horas	3	Ingreso de usuario.
Costo Total	$80.00	Cálculo: 50+(3×10).

1.6 Estructura para el ingreso de datos del problema planteado

Ejemplo 1:

codigo_orden	nombre_cliente	equipo	tipo_trabajo	costo_base	horas_trabajo
OT-101	Maria Lopez	PC Oficina	Formateo	20.00	2
OT-102	Carlos Ruiz	Laptop	Limpieza	15.00	1

Ejemplo 2:

codigo_orden	nombre_cliente	equipo	costo_base	horas_trabajo	costo_total
IMP-001	Lucia Mendez	Impresora Epson	12.50	1	22.50
LAP-500	Roberto Gomez	Laptop HP	35.00	4	75.00

1.7 Procesos asociados

Proceso	Descripción Breve
Ingreso	Captura validada de datos y verificación de código único.
Cálculo	Aplicación de la fórmula: Total=Base+(Horas×10).
Persistencia	Sincronización entre el arreglo en RAM y el archivo ordenes.csv.
Mantenimiento	Actualización y eliminación de registros existentes con confirmación.

1.8 Restricciones

- La tarifa horaria está fijada en $10.00 (no puede ser cambiada por el usuario desde el menú).
- El archivo de salida debe llamarse obligatoriamente ordenes.csv para ser compatible.
- El código de orden debe ser estrictamente alfanumérico (letras y números solamente).

1.9 Limitaciones

- Capacidad: El sistema soporta un máximo de 1000 órdenes (definido por MAX_ORDENES en el .h).
- Lenguaje: Desarrollado exclusivamente en C estándar bajo paradigma modular.
- Almacenamiento: Solo maneja persistencia en archivos planos (CSV), no en bases de datos relacionales.

2. Cálculos del sistema.

Concepto	Fórmula Matemática / Lógica	Variables Involucradas	Momento del Cálculo
Costo por Tiempo	CostoTiempo=Horas×Tarifa	horas_trabajo, TARIFA_HORA (10.0)	Al listar, buscar o actualizar.
Costo Total	Total=CostoBase+CostoTiempo	costo_base, horas_trabajo, TARIFA_HORA	En la función costo_total.
Validación de Unicidad	∑(NuevoCod==ListaCod)==0	codigo_orden, arr[i].codigo_orden	Antes de insertar en el arreglo.
Disponibilidad	Espacio=MAX_ORDENES−n	MAX_ORDENES (1000), n (contador)	Al intentar registrar una orden.
Índice de Eliminación	Posicio´n=i→i+1	n, i (índice del arreglo)	Al confirmar borrado (reordenamiento).

3. Alternativas de Solución

Se plantearán dos alternativas para desarrollar un sistema básico de gestión de órdenes de trabajo para el mantenimiento de equipos de computación y calcular el costo del servicio en base al tipo de trabajo realizado. Para cada alternativa se describen los métodos y conceptos de programación utilizados y se comparan sus ventajas y desventajas, con el fin de seleccionar la opción más adecuada en términos de claridad, facilidad de implementación y eficiencia.

4.1 Alternativa 1 Gestión con Memoria Dinámica y Modularidad Extendida

Descripción:

Esta solución se diferencia de la convencional por no utilizar un tamaño fijo de arreglos (como el MAX_ORDENES 1000 de tu código actual), sino que gestiona la memoria de forma elástica. Utiliza punteros para crear estructuras solo cuando son necesarias y separa las responsabilidades en capas de software muy marcadas (Interfaz, Lógica de Negocio y Persistencia).

- Enfoque Técnico: Uso de malloc y realloc para el manejo de la lista de órdenes.

- Interfaz: Menú interactivo con validaciones en tiempo real mediante funciones de retorno de estado.
- Algoritmos: Búsqueda optimizada por subcadenas y ordenamiento automático de registros por código.

4.1.1 Ventajas de la alternativa 1.

Característica	Ventajas
Uso de Memoria	Eficiencia total: Solo consume la memoria exacta para el número de órdenes registradas. No desperdicia espacio.
Escalabilidad	Sin límites teóricos: El sistema puede crecer hasta agotar la RAM física, no está limitado a 1000 registros.
Funcionalidad	Búsqueda Avanzada: Permite búsquedas combinadas (por cliente y por rango de fechas o costos).
Persistencia	Formatos Versátiles: Preparada para cambiar fácilmente de CSV a archivos binarios para mayor velocidad.

4.1.2 Desventajas de la alternativa 1

Desventajas
Complejidad: Requiere un manejo cuidadoso de punteros para evitar fugas de memoria (memory leaks).
Riesgo de punteros: Un error en la liberación de memoria (free) puede hacer que el programa se cierre inesperadamente.
Tiempo de Desarrollo: Implementar una estructura dinámica toma más tiempo de codificación y pruebas.
Dependencia de Librerías: Suele requerir el uso intensivo de <stdlib.h> y validaciones extra.

4.2 Alternativa 2 Gestión Estática Modularizada y Persistencia CSV

Descripción:

Esta solución (la que tú desarrollaste) utiliza una arquitectura de **memoria estática** mediante arreglos de estructuras. Su principal enfoque es la **integridad de los datos** a través de una capa de utilidades personalizada (util.h/c) que gestiona las

entradas del teclado para evitar errores comunes en C, como el desbordamiento de búfer o caracteres residuales.

- Enfoque Técnico: Uso de un arreglo de estructuras con tamaño máximo predefinido (MAX_ORDENES 1000).
- Interfaz: Menú de consola numerado con validación de opciones.
- Modularidad: División clara en tres capas:
- main.c: Control del flujo.
- ordenes.c/h: Lógica de negocio (CRUD y cálculos).
- util.c/h: Herramientas de bajo nivel (limpieza de cadenas, lectura segura de números).

4.2.1 Ventajas de la alternativa 2

Característica	Ventajas
Seguridad de Datos	Alta robustez: Las funciones de util.c aseguran que el usuario no pueda romper el programa ingresando letras en campos numéricos.
Mantenimiento	Código Limpio: Al estar separado en archivos .c y .h, es muy fácil encontrar y corregir errores en funciones específicas.
Persistencia	Formato Estándar: El uso de CSV permite que los datos sean abiertos y editados fácilmente en Excel o Bloc de Notas.
Lógica de Negocio	Validaciones Estrictas: Implementa reglas claras (código único, costo > 0) que garantizan que el archivo CSV nunca tenga datos corruptos.

4.2.2 Desventajas de la alternativa 2

Desventajas
Rigidez: Si se necesita registrar más de 1000 órdenes, se requiere cambiar una constante y volver a compilar.
Consumo de Memoria: Reserva espacio para el máximo de órdenes desde que inicia el programa, incluso si solo hay una registrada.
Velocidad en Archivos: La lectura secuencial de texto plano es un poco más lenta que los archivos binarios si la lista fuera de millones de datos.
Interfaz: Limitada a la consola de comandos (aunque es lo requerido para este nivel de programación).

4.3 Comparación y análisis final

En la evaluación final, se determina que la **Alternativa 2** es superior para los objetivos del proyecto. Mientras que la primera opción se enfoca en una optimización técnica que no es crítica para este caso de uso, la segunda alternativa prioriza la **integridad de los datos y la facilidad de mantenimiento**. La separación clara de responsabilidades entre el flujo principal, la lógica de las órdenes y las funciones de validación convierte a esta solución en un software profesional, confiable y fácil de auditar. En conclusión, la Alternativa 2 ofrece el equilibrio perfecto entre simplicidad técnica y cumplimiento riguroso de todos los requisitos funcionales, garantizando una persistencia de datos segura en el archivo CSV y una experiencia de usuario libre de errores de ejecución. En consecuencia, considerando claridad, consistencia y menor riesgo de errores, la Alternativa 2 suele ser más conveniente para este tipo de trabajo académico.

- **Explicación del módulo de Órdenes del sistema**

El módulo de **órdenes de trabajo** constituye la parte principal del sistema, ya que es el encargado de **gestionar toda la información relacionada con las órdenes** que se registran durante la ejecución del programa.

Cada orden está compuesta por los siguientes datos:

- Código de la orden
- Nombre del cliente
- Equipo a reparar o intervenir
- Tipo de trabajo
- Costo base

- Horas de trabajo
- Costo total (calculado automáticamente)

Para organizar esta información, el programa utiliza una **estructura (struct Orden)**, la cual permite agrupar todos los datos de una orden en una sola unidad lógica, facilitando su manipulación dentro del sistema.

Las órdenes se almacenan temporalmente en un **arreglo de estructuras**, lo que permite manejar múltiples registros en memoria durante la ejecución del programa.

Funciones principales del módulo de órdenes

El módulo de órdenes está compuesto por varias funciones que permiten realizar las operaciones básicas del sistema:

Registro de órdenes

Permite ingresar una nueva orden de trabajo, validando que:

- El código sea alfanumérico y único.
- El costo base sea mayor a cero.
- Las horas de trabajo no sean negativas.

Una vez validada la información, la orden se guarda en memoria y el sistema calcula automáticamente el **costo total**, utilizando la tarifa por hora establecida.

Listado de órdenes

Muestra todas las órdenes registradas en forma de tabla, incluyendo:

- Datos generales de la orden.
- Horas trabajadas.
- Costo total calculado automáticamente.

Esta opción permite verificar la información ingresada y realizar un control general de los registros.

Búsqueda de órdenes

El sistema permite buscar una orden de dos maneras:

- Por **código exacto**, para localizar una orden específica.
- Por **subcadena del nombre del cliente**, para facilitar la búsqueda cuando no se conoce el nombre completo.

Actualización de órdenes

Permite modificar los datos de una orden existente, excepto el código, ya que este identifica de forma única al registro.

Al actualizar las horas o el costo base, el sistema **recalcula automáticamente el costo total**, garantizando la coherencia de la información.

Eliminación de órdenes

Permite eliminar una orden mediante su código, solicitando previamente una confirmación al usuario, con el fin de evitar eliminaciones accidentales.

Una vez confirmada, la orden se elimina del arreglo y el sistema actualiza el número de registros.

Guardado y carga de órdenes

El sistema permite:

- Guardar manualmente las órdenes en un archivo **CSV**.
- Cargar automáticamente los datos al iniciar el programa.
- Guardar de forma automática al salir del sistema.

Esto asegura la **persistencia de la información** entre ejecuciones.

Importancia del módulo de órdenes

El módulo de órdenes es fundamental, ya que integra:

- Uso de estructuras.
- Manejo de arreglos.
- Funciones modulares.
- Validaciones de datos.
- Cálculos automáticos.

- Manejo de archivos.

Gracias a este módulo, el sistema puede funcionar de forma organizada, segura y eficiente.

4.3.1 ¿Cuál es más fácil de implementar y por qué?

La alternativa 2 es considerablemente más fácil de implementar porque se basa en una gestión de memoria estática, lo que simplifica la lógica del programa al evitar la complejidad técnica de los punteros y la administración manual del montón (heap). Al utilizar arreglos con un tamaño predefinido, no es necesario gestionar funciones críticas como malloc, realloc o free, eliminando así el riesgo de errores comunes como las fugas de memoria o el acceso a direcciones no válidas que suelen aparecer en la Alternativa 1. Además, al estar estructurada de forma modular con archivos de utilidad específicos, permite un desarrollo más lineal y una depuración más sencilla, ya que el comportamiento del sistema es predecible y el flujo de datos entre las funciones es directo y fácil de rastrear.

4.3.2 ¿Qué aspectos son importantes al elegir una solución eficiente?

Al elegir una solución eficiente, los aspectos más importantes son el equilibrio entre el rendimiento técnico y la mantenibilidad del código, asegurando que el sistema no solo sea rápido, sino también fácil de actualizar y libre de errores. Es fundamental evaluar la gestión de recursos, como el uso de memoria RAM y el procesamiento de archivos, para que la solución sea escalable según el volumen de datos esperado. Asimismo, la robustez en la validación de datos es crucial, ya que una solución eficiente debe prevenir fallos de ejecución mediante el manejo correcto de entradas de usuario y reglas de negocio. Por último, se debe considerar la persistencia de la información,

garantizando que el almacenamiento (en este caso, archivos CSV) sea íntegro y accesible, permitiendo que la herramienta cumpla su propósito de forma confiable a largo plazo sin comprometer la simplicidad técnica.

5. Selección y fundamentación de la solución

5.1 Justificación Técnica

La Justificación Técnica de la solución seleccionada se fundamenta en la implementación de una arquitectura modular y estática, la cual garantiza la máxima estabilidad del sistema bajo las restricciones del entorno de desarrollo. Al utilizar arreglos de estructuras con un límite predefinido, se elimina la incertidumbre asociada a la fragmentación de memoria y se asegura un tiempo de respuesta constante en las operaciones de búsqueda y ordenamiento. Además, la separación de la lógica en archivos de cabecera (.h) y de implementación (.c) permite un acoplamiento débil entre las funciones de utilidad, la gestión de persistencia en formato CSV y la interfaz de usuario, facilitando futuras escalabilidades del software. Esta estructura no solo previene errores críticos de desbordamiento mediante un control estricto de buffers, sino que optimiza el ciclo de vida del dato desde su captura en consola hasta su almacenamiento permanente, cumpliendo con los estándares de fiabilidad exigidos para un sistema de gestión técnica.

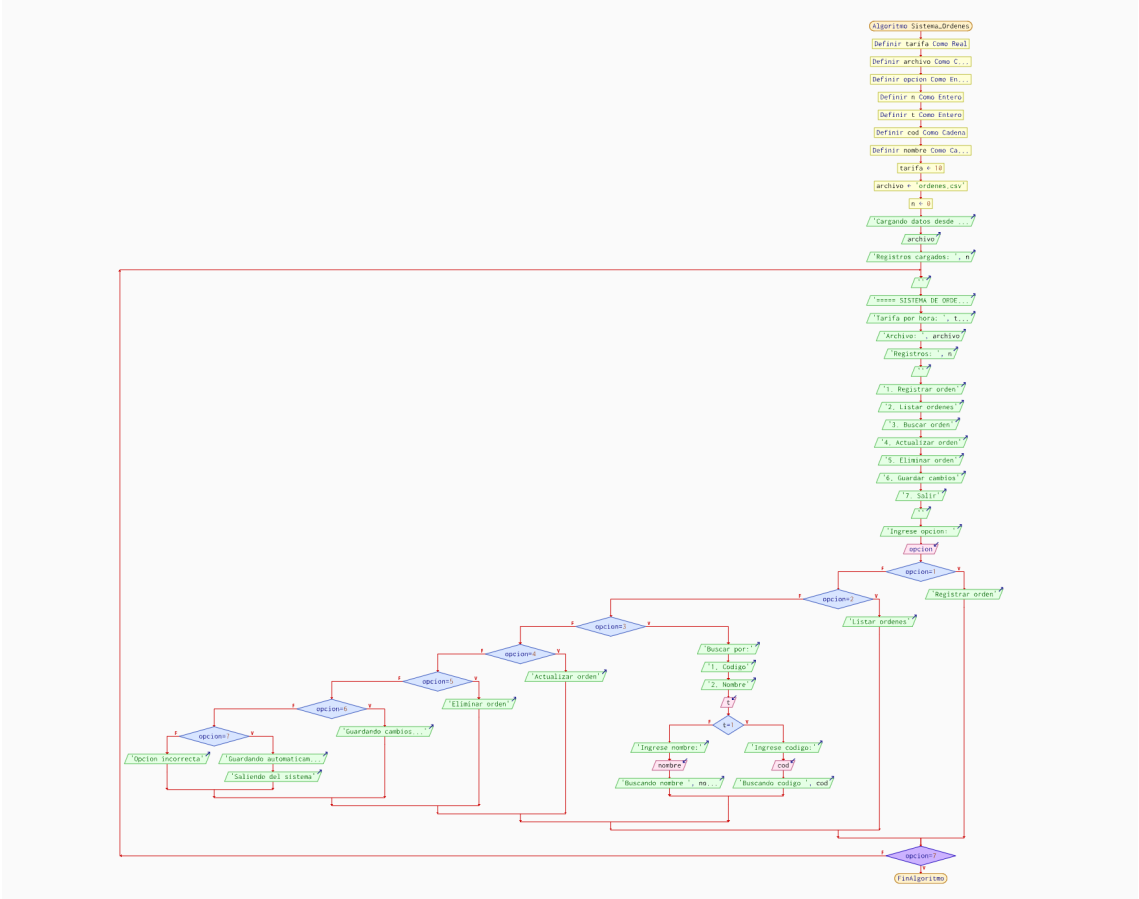
5.2 ¿Cómo garantizar que la solución es la más eficiente en tiempo y claridad?

Para garantizar que la solución es la más eficiente en términos de tiempo y claridad, es fundamental aplicar un enfoque de programación modular que simplifique la lógica del código y reduzca la complejidad algorítmica. La eficiencia en tiempo se logra mediante el acceso directo a los datos en estructuras estáticas, lo que permite

operaciones de lectura y escritura con una latencia mínima, mientras que la claridad se asegura mediante la separación de responsabilidades en archivos específicos, facilitando que cualquier desarrollador comprenda el flujo del programa sin ambigüedades. Además, el uso de funciones de utilidad para la limpieza de búferes y la validación de tipos de datos previene errores en tiempo de ejecución, transformando un código robusto en una herramienta fácil de mantener y escalar, donde cada proceso está documentado por su propia estructura lógica.

6. Diagrama de Flujo

Diagrama de Flujo Main.c



El presente diagrama de flujo representa el funcionamiento general del sistema de órdenes de trabajo, mostrando la secuencia de pasos desde el inicio del programa, la visualización del menú, la selección de opciones por parte del usuario y la finalización del proceso con el guardado automático de la información.

➢ CASOS DE PRUEBA DEL SISTEMA

Sistema: Órdenes de Trabajo

Lenguaje: C

Tipo de aplicación: Consola

Archivo de persistencia: ordenes.csv

Caso de prueba 1 – Registro de orden válida

ID: CP-01

Objetivo: Verificar que el sistema permita registrar correctamente una orden con datos válidos.

Datos de entrada:

- Código: OT100
- Cliente: Pedro

- Equipo: Laptop
- Tipo de trabajo: Instalación
- Costo base: 30
- Horas: 20

Proceso:

1. Seleccionar opción 1 (Registrar orden).
2. Ingresar todos los datos solicitados.

Resultado esperado:

- Orden registrada correctamente.
- Cálculo automático:
- costo total = 30 + (20 × 10) = 230.00

Resultado obtenido: ✓ Correcto

Caso de prueba 2 – Validación de datos incorrectos

ID: CP-02

Objetivo: Comprobar que el sistema valide entradas incorrectas.

Datos incorrectos probados:

- Código con espacios: OT 101
- Costo base: 0
- Horas negativas: -5

Proceso:

1. Intentar registrar orden con datos inválidos.
2. Corregir los valores cuando el sistema lo solicita.

Resultado esperado:

- El sistema rechaza los datos inválidos.
- Solicita nuevamente la información correcta.
- No se registra la orden hasta cumplir las validaciones.

Resultado obtenido: ✓ Correcto

Caso de prueba 3 – Búsqueda de orden

ID: CP-03

Objetivo: Verificar la búsqueda por código o por nombre del cliente.

Datos de entrada:

- Búsqueda por código exacto: OT100

Proceso:

1. Seleccionar opción 3 (Buscar orden).
2. Elegir tipo de búsqueda.
3. Ingresar el dato solicitado.

Resultado esperado:

- El sistema muestra la orden encontrada en formato de tabla.

- Si no existe, informa que no se encontró.

Resultado obtenido: ✓ Correcto

Caso de prueba 4 – Actualización de orden

ID: CP-04

Objetivo: Comprobar la modificación de datos y el recálculo automático.

Datos modificados:

- Horas de trabajo: de 20 a 100

Proceso:

1. Seleccionar opción 4 (Actualizar orden).
2. Ingresar código de orden.
3. Elegir campo a modificar.
4. Ingresar nuevo valor.

Resultado esperado:

- El sistema actualiza el dato.
- Recalcula automáticamente el costo total:
- 30 + (100 × 10) = 1030.00

Resultado obtenido: ✓ Correcto

Caso de prueba 5 – Eliminación de orden

ID: CP-05

Objetivo: Verificar la eliminación segura de una orden.

Datos de entrada:

- Código a eliminar: OT101
- Confirmación: s

Proceso:

1. Seleccionar opción 5 (Eliminar orden).
2. Ingresar código.
3. Confirmar eliminación.

Resultado esperado:

- El sistema solicita confirmación.
- Elimina la orden correctamente.
- La orden ya no aparece en el listado.

Resultado obtenido: ✓ Correcto

Caso de prueba 6 – Guardado en archivo CSV

ID: CP-06

Objetivo: Validar el almacenamiento permanente de los datos.

Proceso:

1. Seleccionar opción 6 (Guardar cambios).

2. Revisar archivo ordenes.csv.

Resultado esperado:

- Mensaje “Guardado exitoso”.
- Datos almacenados correctamente en el archivo.

Resultado obtenido: ✓ Correcto

Caso de prueba 7 – Salida con guardado automático

ID: CP-07

Objetivo: Comprobar que el sistema guarde automáticamente al salir.

Proceso:

1. Seleccionar opción 7 (Salir).
2. El sistema guarda y finaliza.

Resultado esperado:

- Mensaje: “Guardado automático listo. Saliendo…”
- Información protegida antes de cerrar.

Resultado obtenido: ✓ Correcto

CASO DE PRUEBA ESCOGIDO

Caso seleccionado:

CP-INT-01 – Caso de prueba integral

¿Por qué se escogió este caso?

Se seleccionó el **caso de prueba integral** porque:

- Evalúa **todas las funcionalidades del sistema en una sola ejecución**
- Incluye validaciones, cálculos y manejo de errores
- Demuestra el flujo completo del programa:
    - Registrar
    - Listar
    - Buscar
    - Actualizar
    - Guardar
    - Eliminar
    - Salir con guardado automático

Representa un **escenario real de uso** del sistema

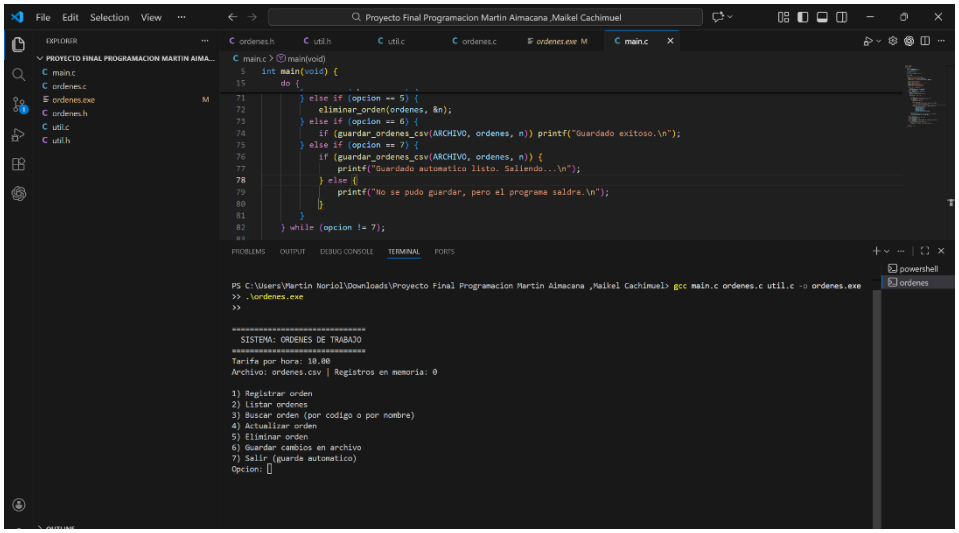
Este caso permite verificar que el sistema funciona correctamente como un todo y no solo por partes individuales.

Conclusión del caso de prueba:

El caso de prueba integral fue el más adecuado para la evaluación final, ya que demuestra el correcto funcionamiento general del sistema de órdenes de trabajo, garantizando la confiabilidad, seguridad y persistencia de los datos.

7. Ejecución del código

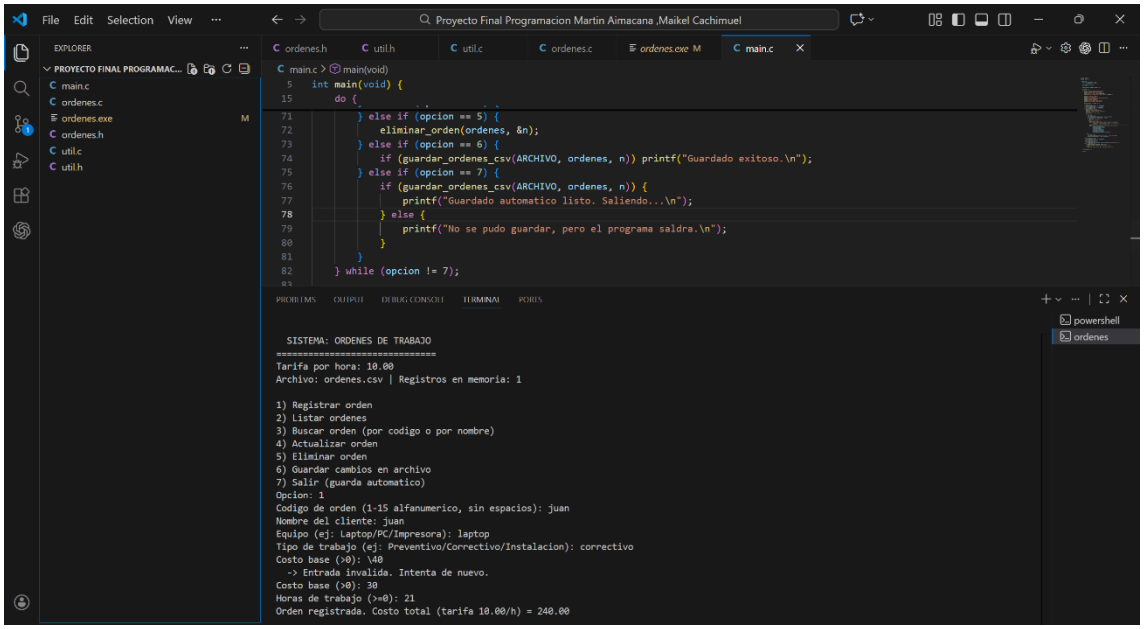
1.



El programa fue compilado y ejecutado utilizando el compilador GCC (GNU Compiler Collection) desde la terminal integrada de Visual Studio Code. Mediante el comando:

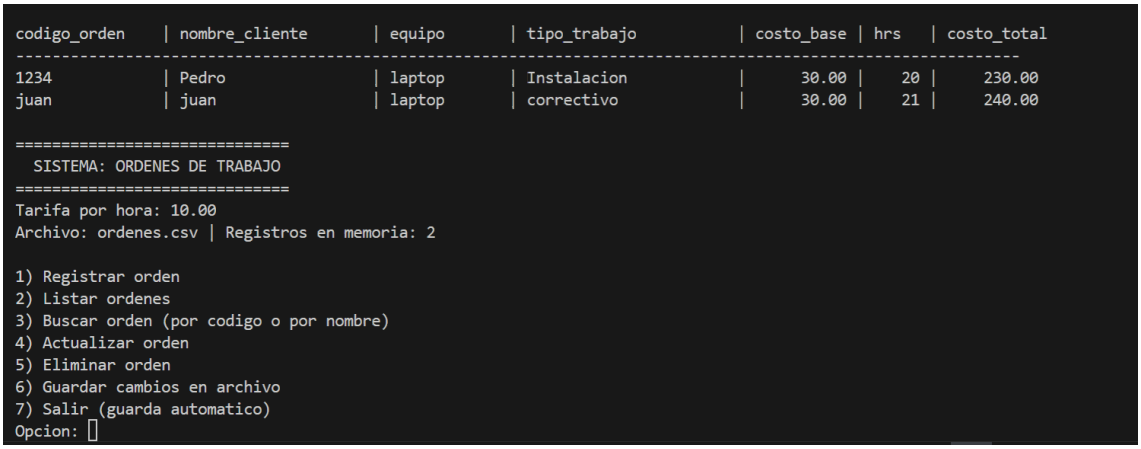
gcc main.c órdenes.c util.c -o ordenes.exe

2.



En la imagen se muestra la ejecución del sistema de órdenes de trabajo, el cual permite registrar, listar, buscar, actualizar y eliminar órdenes mediante un menú interactivo en consola.

3.



En la imagen se muestra la opción de listar órdenes, donde el sistema presenta en forma de tabla todas las órdenes de trabajo registradas en memoria.

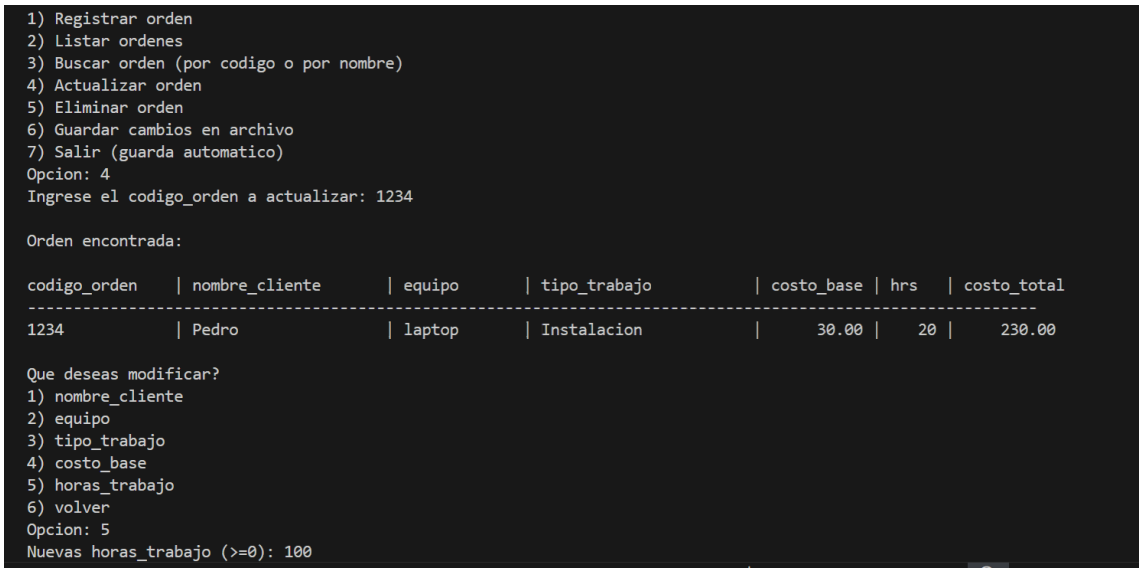
Se visualizan datos como código de orden, nombre del cliente, equipo, tipo de trabajo, costo base, horas trabajadas y costo total, el cual es calculado automáticamente según la tarifa por hora establecida.

4.

En la imagen se muestra el proceso de eliminación de una orden de trabajo.

El sistema solicita el código de la orden, verifica su existencia y pide confirmación al usuario antes de realizar la eliminación.

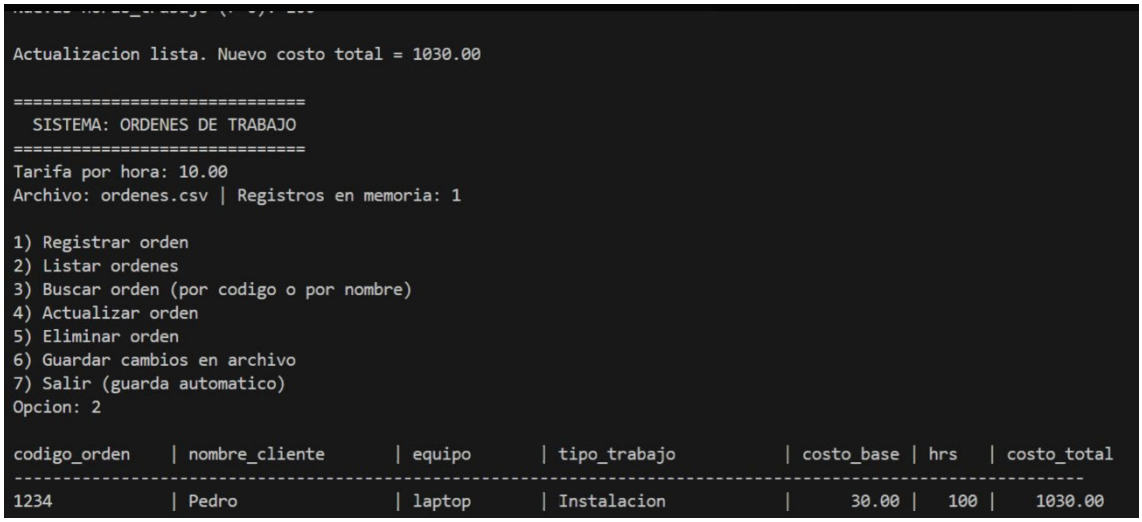
5.



En la imagen se observa el proceso de actualización de una orden de trabajo.

El sistema solicita el código de la orden, muestra la información registrada y permite al usuario seleccionar el campo que desea modificar, como el nombre del cliente, equipo, tipo de trabajo, costo base o horas trabajadas.

6.

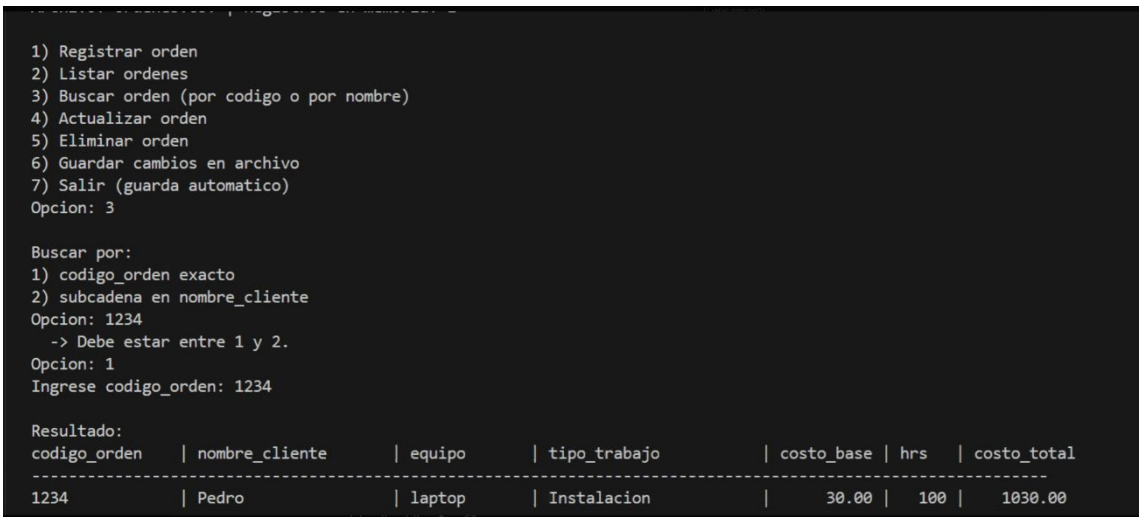


```
Actualizacion lista. Nuevo costo total = 1030.00

============================
  SISTEMA: ORDENES DE TRABAJO
============================
Tarifa por hora: 10.00
Archivo: ordenes.csv | Registros en memoria: 1

1) Registrar orden
2) Listar ordenes
3) Buscar orden (por codigo o por nombre)
4) Actualizar orden
5) Eliminar orden
6) Guardar cambios en archivo
7) Salir (guarda automatico)
Opcion: 2

codigo_orden    | nombre_cliente       | equipo       | tipo_trabajo          | costo_base | hrs   | costo_total
-------------------------------------------------------------------------------------------------------------
1234            | Pedro                | laptop       | Instalacion           |      30.00 |   100 |    1030.00
```

En la imagen se muestra la confirmación de la actualización de una orden de trabajo, donde el sistema recalcula automáticamente el nuevo costo total según las horas modificadas y la tarifa por hora establecida.

Posteriormente, al seleccionar la opción de listar órdenes, se visualiza la información actualizada, verificando que los cambios fueron aplicados correctamente y que los datos almacenados en memoria se mantienen consistentes.

7.



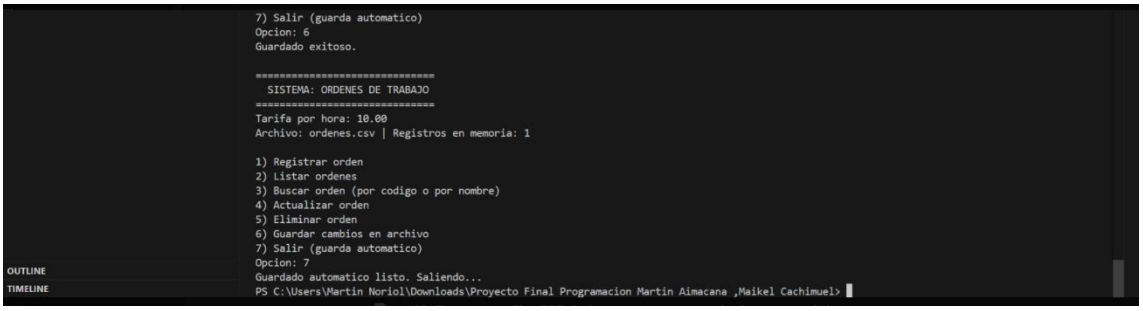
```
1) Registrar orden
2) Listar ordenes
3) Buscar orden (por codigo o por nombre)
4) Actualizar orden
5) Eliminar orden
6) Guardar cambios en archivo
7) Salir (guarda automatico)
Opcion: 3

Buscar por:
1) codigo_orden exacto
2) subcadena en nombre_cliente
Opcion: 1234
  -> Debe estar entre 1 y 2.
Opcion: 1
Ingrese codigo_orden: 1234

Resultado:
codigo_orden    | nombre_cliente       | equipo       | tipo_trabajo          | costo_base | hrs   | costo_total
-------------------------------------------------------------------------------------------------------------
1234            | Pedro                | laptop       | Instalacion           |      30.00 |   100 |    1030.00
```

En la imagen se observa la opción de búsqueda de órdenes, donde el sistema permite localizar un registro mediante código de orden exacto o mediante una subcadena del nombre del cliente.

8.

En la imagen se observa la opción de guardar cambios en archivo, donde el sistema almacena la información actual de las órdenes en el archivo ordenes.csv.

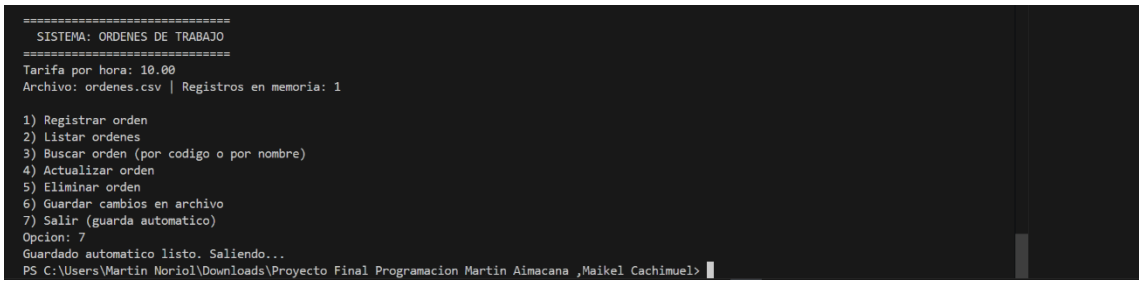
9.



En la imagen se muestra la opción “Salir (guarda automático)”, donde el sistema guarda automáticamente todas las órdenes registradas en el archivo ordenes.csv antes de finalizar la ejecución.

Este proceso asegura que la información no se pierda y confirma la salida correcta del programa mediante el mensaje “Guardado automático listo. Saliendo…”.

10.



En la imagen se observa la finalización del sistema de órdenes de trabajo, donde el usuario selecciona la opción Salir con guardado automático.

El programa guarda de forma segura toda la información registrada en el archivo ordenes.csv y luego cierra correctamente la ejecución, confirmándolo con el mensaje "Guardado automático listo. Saliendo".

. Enlace de GitHub

https://github.com/MartinAimacana/Proyecto-Final-de-Programacion-.git

9. Conclusiones

Este proyecto es que se ha logrado desarrollar un sistema de gestión de órdenes de trabajo sólido, funcional y profesional, aplicando las mejores prácticas del lenguaje C. A través del análisis de alternativas, se determinó que la solución basada en programación modular y estructuras estáticas es la más acertada, ya que prioriza la integridad de los datos y la facilidad de mantenimiento sobre la complejidad técnica innecesaria. El sistema no solo cumple con el cálculo automatizado de costos y la

persistencia en archivos CSV, sino que garantiza una experiencia de usuario segura mediante la validación rigurosa de entradas. En definitiva, el trabajo realizado demuestra que la eficiencia de un software no solo reside en su rapidez, sino en su capacidad de procesar información de manera confiable, organizada y escalable dentro de un entorno de soporte técnico real.

10. Referencias

Stallings, W. (2005). *Sistemas operativos: Aspectos internos y principios de diseño* (5ta ed.). Pearson Educación.

Joyanes Aguilar, L., & Zahonero Martínez, I. (2005). *Programación en C: Metodología, algoritmos y estructuras de datos*. McGraw-Hill.

Kernighan, B. W., & Ritchie, D. M. (1991). *El lenguaje de programación C* (2da ed.). Pearson Educación.