UNIVERSIDAD ADOLFO IBÁÑEZ

Tarea 1

Pizzería en C

Integrantes: Sofía Saez, Constanza Lucero y Valentina Meyer

Curso: Lenguajes y Paradigmas de Programación

Profesor: Justo Vargas

Fecha: 05-04-2025

I. Introducción

En esta tarea, desarrollamos un programa en lenguaje C que simula un sistema de análisis de ventas para una pizzería ficticia. A partir de un archivo CSV con datos de ventas, el programa debía ser capaz de calcular diversas métricas.

El enfoque principal fue aplicar conceptos claves de programación procedural como el uso de estructuras (struct), lectura de archivos, punteros, y en especial, punteros a funciones. La idea era que el sistema pudiera recibir comandos desde la consola con distintas métricas y entregar un correcto resultado.

II. Objetivo

El objetivo principal de esta tarea fue desarrollar un programa modular en C que pudiera analizar un archivo CSV con datos de ventas de pizzas, calculando métricas como la pizza más vendida, la fecha con más ingresos, el ingrediente más utilizado, entre otras. Para eso, organizamos el código en distintos archivos: *main.c* para la ejecución principal, *metricas.c* para las funciones de cálculo, y *structs.h* para definir la estructura Venta, que usamos como base para guardar cada línea del CSV en un registro estructurado.

Decidimos estructurarlo de esta manera para mantener una separación clara entre la lógica del programa, las funciones específicas de métricas, y la definición de datos, lo que hizo más fácil el trabajo en equipo y el realizar modificaciones. Todas las métricas se implementaron como funciones con la misma firma y se almacenaron en un arreglo de punteros a funciones, lo que nos permitió ejecutar dinámicamente las métricas solicitadas desde los argumentos de consola.

Además de aprender el uso de struct, fgets, strtok, manipulación de archivos y trabajar con punteros a funciones, también usamos herramientas como make para automatizar la compilación y GitHub para organizar el trabajo colaborativo y mantener control de versiones del proyecto.

III. Arquitectura del proyecto

El proyecto se organizó siguiendo una estructura modular para facilitar su comprensión, mantenimiento y escalabilidad. A continuación, se presenta la estructura de archivos que componen la aplicación:

- main.c: Controla el flujo principal de ejecución del programa.
- metricas.c: Contiene funciones especializadas para el análisis y cálculo de métricas.
- metricas.h: Declara las funciones públicas del módulo de métricas.
- structs.h: Define la estructura de datos usada para representar cada venta.
- Makefile: Automatiza la compilación del proyecto.
- ventas.csv: Archivo con datos de las órdenes de clientes.
- README.md: Documentación general del proyecto.

IV. Estructuras de Datos

Para representar las órdenes de compra, se diseñó la siguiente estructura:

```
typedef struct {
    char* pizza_name;
    int quantity;
    char* order_date;
    float total_price;
} Venta;
```

Las órdenes se almacenan en un arreglo de estructuras Venta. Esto permite recorrerlas eficientemente para realizar análisis posteriores.

V. Estrategia de Lectura del Archivo CSV

La lectura del archivo ventas.csv se realiza utilizando funciones estándar de C, como fgets() para obtener cada línea del archivo, y strtok() para dividir las líneas en campos delimitados por comas.

El flujo de lectura incluye:

- Validación de cada línea para verificar que contenga el número correcto de campos.
- Conversión de los campos al tipo de dato correspondiente (int, float, char*).
- Almacenamiento dinámico de los datos leídos en memoria.

VI. Uso de Punteros a Funciones

Para facilitar la ejecución modular de diversas métricas, se implementó un sistema basado en punteros a funciones:

typedef void (*MetricFunction)(Order*, int);

También, una función principal (aplicar_metricas) recibe un arreglo de punteros a funciones y las ejecuta secuencialmente:

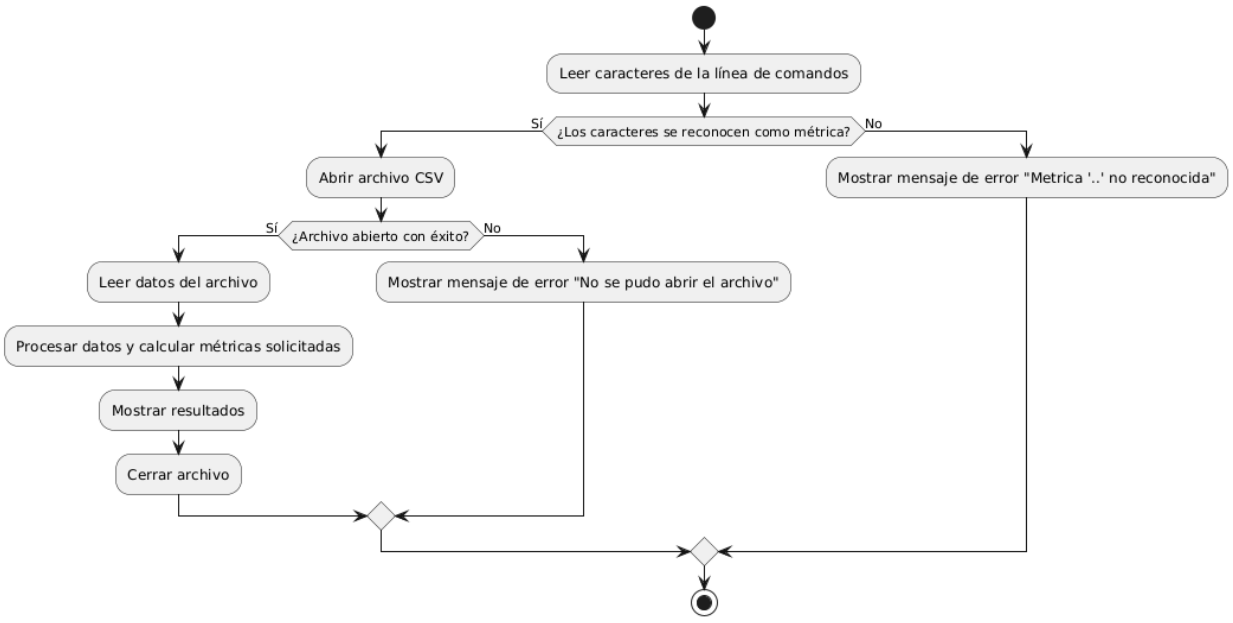
```
void aplicar_metricas(Order* ordenes, int n, MetricFunction* funciones, int
cantidad_funciones) {

        for (int i = 0; i < cantidad_funciones; ++i) {

            funciones[i](ordenes, n);

        }
    }
```

VII. Diagrama de Flujo



VIII. Reflexión

Al inicio de la realización de la tarea estábamos complicadas para abordar lo que se nos pedía. Aunque habíamos visto punteros en clases, no entendíamos del todo cómo funcionaban los punteros a funciones ni cómo aplicarlos en un proyecto real. Nos costó un poco comprender cómo hacer que el programa eligiera qué métrica ejecutar a partir de los argumentos entregados por consola.

También nos costó entender cómo organizar bien el código en varios archivos, ya que al principio teníamos todo junto en main.c. Separarlo en archivos como metricas.c y structs.h fue clave para que el proyecto fuera más ordenado y para poder avanzar de forma más clara.

Otra dificultad fue manejar correctamente los strings en C, especialmente al construir los mensajes que devolvía cada función. Tuvimos errores con cadenas mal formateadas o cortadas, y también con la impresión en consola. En ese punto, y en otros momentos del desarrollo, nos apoyamos en Chat GPT para entender por qué ocurrían ciertos errores y cómo solucionarlos. Le mostramos partes de nuestro código y la IA nos explicaba posibles causas y formas de corregirlas.

Muchas veces también nos ayudó a interpretar errores que nos salían en la consola, o a entender conceptos que no nos quedaban claros solo con la documentación. Por ejemplo, entender bien cómo usar *snprintf*, cómo trabajar con strtok, o cómo inicializar bien los arreglos de punteros a funciones. Aun así, cada solución la validamos nosotras mismas haciendo pruebas, leyendo el código y adaptándolo si era necesario.

A pesar de las dificultades, sentimos que aprendimos harto. Pudimos aplicar cosas que solo habíamos visto de forma teórica y ahora entendemos mucho mejor para qué sirven los punteros a funciones y cómo programar de manera más estructurada. También fue valioso aprender a usar herramientas como make y GitHub, que son cosas que probablemente vamos a seguir usando en otros ramos o proyectos más grandes.

IX. Uso de IA

Para el desarrollo de la tarea nos apoyamos en Chat GPT para diversas dudas o errores que nos iban surgiendo a medida que avanzamos en el desarrollo del código.

En primer lugar, le pedimos a Chat GPT que nos hiciera una especie de código general para tomarlo como modelo a seguir en base a lo que se nos solicitaba, de esta forma poder guiarnos sobre la estructura que necesitábamos, funciones más específicas que no conocíamos y nos simplifican el desarrollo del programa. Una vez con este modelo generado lo fuimos adaptando completamente a lo que necesitábamos y mejorando en caso de ser necesario.

Una de los primeros usos que le dimos a la IA fue entender de mejor manera cómo usar correctamente los punteros a funciones para poder lograr nuestro objetivo en específico, pues si bien lo vimos en clases nos costó llevarlo a la práctica.

Por otra parte, hicimos uso de Chat GPT para saber cuál era la mejor forma de hacer que leyera el archivo CSV, pues hasta entonces no sabíamos lo importante que era el proceso de leer los datos que estaban como texto y transformarlos a estructuras que el programa pudiese leer.

Otro problema que nos ayudó a resolver fue en las funciones que devuelven las respuestas a la métricas solicitadas, nos pasó que en vez de devolver la pizza más vendida el programa devolvía los ingredientes de esta, finalmente, gracias a Chat GPT logramos entender de donde provenía el error y posteriormente corregirlo.

Es importante mencionar que las respuestas que nos entregaba la IA las validamos con conocimientos aprendidos en clase si era el caso o buscando más información al respecto para lograr entender completamente su diseño y posteriormente adaptarlas a lo que necesitábamos o teníamos.

En conclusión, el uso de la IA fue un apoyo importante para responder dudas, verificar si íbamos por el camino correcto o simplemente buscar orientación durante el transcurso de la tarea.

X. Limitaciones y posibles mejoras

En nuestro programa, estamos trabajando con un arreglo de estructuras para almacenar información de ventas. Inicialmente intentamos aumentar el límite de órdenes a 10000, pero al compilar y ejecutar el programa, apareció un error de "segmentation fault". Este error ocurre cuando el programa intenta acceder a una parte de la memoria que no tiene permiso de usar.

Esto sucede porque el arreglo ventas[] que utilizamos para guardar los datos se declara en la pila de memoria, la cual tiene un tamaño limitado. Al intentar reservar espacio para 10000 elementos, se excede esa capacidad, lo que lleva al fallo en tiempo de ejecución.

Por esta razón, decidimos establecer un límite de 1000 órdenes, un número que asegura que el programa funcione de forma estable sin sobrepasar la memoria disponible. Esta es una medida de prevención para evitar errores de acceso indebido a memoria y garantizar el correcto funcionamiento del sistema.

A futuro, nos gustaría mejorar el manejo de memoria del programa para poder procesar una mayor cantidad de órdenes sin estar limitadas por el tamaño de la pila. Una posible mejora sería implementar asignación dinámica de memoria utilizando malloc, lo que permitiría adaptar el tamaño del arreglo de ventas según la cantidad de datos en el archivo CSV.

Esto haría que el programa fuera más flexible, escalable y capaz de manejar volúmenes mayores de información, permitiendo trabajar con archivos más grandes sin comprometer la estabilidad del sistema. En definitiva, sería un paso importante para hacer la aplicación más robusta y preparada para escenarios de uso más exigentes.