Informe App1: Análisis de Ventas de Pizzas

Integrantes: Camilo Bresciani, Vicente Köhler
Sección 2
Profesor: Justo Vargas
Fecha de entrega: 05/04/2025

1. Introducción

El objetivo principal de este proyecto es desarrollar una aplicación en C que lea un archivo CSV que contiene datos sobre ventas de pizzas y calcule diversas métricas relacionadas con las ventas. Entre las métricas implementadas se incluyen: la pizza más vendida, la pizza menos vendida, el día con más ventas en términos de dinero, el día con menos ventas, entre otras. Esta herramienta tiene como propósito ayudar en la toma de decisiones sobre qué pizzas son más populares y qué días generan más ventas.

Total de métricas:

- Pizza más vendida (pms)
- Pizza menos vendida (pls)
- Día con más ventas en términos de dinero (dms)
- Día con menos ventas en términos de dinero (dls)
- Fecha con más pizzas vendidas (dmsp)
- Fecha con menos pizzas vendidas (dlsp)
- Promedio de pizzas por orden (apo)
- Promedio de pizzas por día (apd)
- Ingrediente más vendido (ims)
- Ventas por categoría (hp)

2. Metodología, estructura del proyecto y explicación de archivos

Descripción de los datos: El archivo CSV contiene registros de ventas, donde cada línea describe una venta con los siguientes campos:

- pizza_id: Identificador único de la pizza.
- order_id: Identificador de la orden.
- pizza_name: Nombre de la pizza.
- quantity: Cantidad de pizzas vendidas en la orden.
- order_date: Fecha en la que se realizó la orden.
- order_time: Hora de la orden.
- unit_price: Precio unitario de la pizza.
- total_price: Precio total de la venta.
- pizza_size: Tamaño de la pizza (M, L, etc.).
- pizza_category: Categoría de la pizza (Classic, Veggie, etc.).

- pizza_ingredients: Ingredientes de la pizza.

- Funciones implementadas: Cada métrica fue calculada mediante una función que recorre las órdenes almacenadas en memoria y realiza el análisis correspondiente:
    - **pms:** Calcula la pizza más vendida sumando la cantidad vendida de cada tipo de pizza.
    - **pls:** Calcula la pizza menos vendida, tomando en cuenta las cantidades de cada tipo de pizza.
    - **dms:** Encuentra el día con más ventas sumando los precios de las órdenes realizadas en cada fecha.
    - **dls:** Encuentra el día con menos ventas sumando los precios de las órdenes realizadas en cada fecha.
    - **dmsp:** Encuentra la fecha con más pizzas vendidas, sumando las cantidades por día.
    - **dlsp:** Encuentra la fecha con menos pizzas vendidas.
    - **apo:** Calcula el promedio de pizzas vendidas por orden.
    - **apd:** Calcula el promedio de pizzas vendidas por día.
    - **ims:** Encuentra el ingrediente más vendido, contando la cantidad de veces que aparece en las órdenes.
    - **hp:** Calcula las ventas por categoría, sumando las cantidades vendidas en cada categoría de pizza.

- Estructura: El programa fue estructurado de forma modular para separar responsabilidades. Esta modularidad mejora la claridad del código, permite reutilizar funciones, facilita el trabajo en equipo y hace más sencillo mantener o extender el programa en el futuro.
- **main.c:** Este es el programa principal, el cual toma los argumentos desde consola (ventas.csv y las métricas a calcular), para luego llamar a una función que carga los datos del CSV en memoria (load_orders). Tal como se indicó en la instrucción del trabajo, utiliza punteros a funciones para ejecutar dinámicamente las métricas solicitadas. Finalmente imprime los resultados en pantalla y libera memoria al finalizar.
- **orders.c:** Contiene funciones relacionadas con la carga de datos del archivo CSV. Específicamente la función load_orders, la cual abre el archivo ventas.csv, lo lee línea por línea extrae y convierte los campos necesarios y carga los datos en un arreglo de estructuras order.
- **orders.h:** Define la estructura struct order, que representa una fila del CSV. Incluye campos como quantity, que indica la cantidad de pizzas y order_date, que indica la fecha de la orden. Se pueden agregar otros campos adelante de ser necesario, por lo que es escalable. También declara la función load_orders.
- **metrics.c:** Contiene la implementación de todas las métricas solicitadas en el enunciado (apo, apd, etc). Cada métrica es una función con esta firma: char*

metrica(int *size, struct order *orders), tal como se indica en el enunciado, la cual utiliza punteros a funciones para que puedan llamarse dinámicamente. Luego devuelve una cadena de texto con el resultado (usando malloc).

-
- **metrics.h:** Declara todas las funciones métricas para que puedan ser usadas en main.c. También incluye el #include "orders.h" porque las métricas usan struct order.
- **Makefile:** Es el archivo que automatiza la compilación. Define como compilar todo los ".c" y generar el ejecutable App1. Una vez teniendo el Makefile definido, se puede compilar con solo escribir "make" en la terminal.
- **ventas.csv:** Archivo que contiene todos los datos reales de las ventas de pizza. Es leído por orders.c. De acá las funciones extraen los datos para evaluar las métricas.

Para que el código nos entregue los resultados, se utiliza el siguiente procedimiento dentro del terminal. El primer paso es limpiar el make usando make clean que nos elimina los archivos generados por la compilación que son los archivos .o. Posterior a eso se la función make que compila y construye el proyecto para luego usar " .\app1.exe ventas.csv pms pls dms dls dmsp dlsp apo apd ims hp" que ejecuta el programa app1 utilizando la base de datos ventas.csv y aplicando cada función.

3. Resultados

Para ver si el código funcionaba correctamente en cualquier caso utilizamos unos datos de ejemplo generados aleatoriamente por inteligencia artificial similares a los entregado en el enunciado de la tarea, con los cuales obtuvimos los siguientes resultados:

- Pizza más vendida: "The Hawaiian Pizza (3 unidades)"
- Pizza menos vendida: "The Pepperoni Pizza, The Veggie Pizza (1 unidad)"
- Día con más ventas: "1/1/2015 ($54.00)"
- Día con menos ventas: "1/2/2015 ($13.25)"
- Fecha con más pizzas vendidas: "1/1/2015 (4 pizzas)"
- Fecha con menos pizzas vendidas: "1/2/2015 (1 pizza)"
- Promedio de pizzas por orden: "1.67"
- Promedio de pizzas por día: "2.50"
- Ingrediente más vendido: "Mozzarella Cheese (5 apariciones)"
- Ventas por categoría:
    - Classic: 4
    - Veggie: 1

Los datos de ejemplo utilizados se almacenaron en el archivo ventas.csv. Para verificar que la lectura del archivo CSV fuera correcta, se realizó una verificación manual de los datos, lo cual nos permitió comprobar que las métricas se calculaban correctamente.

Mediante este ejercicio, pudimos ver que los resultados obtenidos son coherentes con los datos de entrada del archivo CSV. Por ejemplo, la pizza más vendida fue "The Hawaiian Pizza", que corresponde con lo esperado, ya que es la pizza con mayor cantidad en el archivo de ventas. Similarmente, las otras métricas como el día con más ventas y el ingrediente más vendido se alinean con los datos que esperábamos, lo que indica que el programa está procesando los datos correctamente.

La métrica promedio de pizzas por orden muestra un valor de 1.67, lo que significa que, en promedio, cada orden contiene aproximadamente 2 pizzas. Este cálculo es adecuado considerando que algunas órdenes son de una sola pizza y otras incluyen más de una.

Por otro lado, la pizza menos vendida muestra dos pizzas (The Pepperoni Pizza y The Veggie Pizza) con el mismo número de unidades vendidas (1), lo que indica que ambas están empatadas en la categoría de "menos vendidas".

Utilizamos este conjunto de datos aleatorios generado por inteligencia artificial para confirmar que nuestro código funciona para cualquier dataset sin problemas. Vimos que los resultados fueron consistentes con los datos reales por lo que pudimos confirmar que funcionaba bien.

4. Conclusiones y reflexión

Este proyecto permitió desarrollar una herramienta que ayuda a analizar las ventas de pizzas mediante métricas clave, lo que puede ser útil para determinar las preferencias de los clientes y las tendencias de ventas. Las métricas implementadas proporcionan información valiosa para optimizar la gestión de inventarios y la planificación de promociones.

Las métricas son fácilmente adaptables para otros contextos, y la aplicación puede extenderse para manejar más tipos de datos o realizar análisis más profundos. Con que solo se almacenen de forma ordenada los datos, en el futuro es posible usar gran cantidad de datos con este mismo código.

En cuanto a la forma de realizar las métricas, en la instrucción se indica que se aceptan múltiples métricas en una misma ejecución, por lo que finalmente, luego de trabajar algunas métricas por separado, decidimos que la mejor opción era calcular y mostrar todas las métricas en una sola ejecución. Comenzamos haciendo las métricas de forma independiente, una por una, lo que funcionó bien. Pero luego tuvimos la idea de hacer un único código que nos proporcione todas las métricas, lo que a nuestro parecer resulta mucho más práctico.

Respecto a lo más complejo de la tarea, ambos concordamos que lo más desafiante fue el uso de GitHub, ya que es primera vez que nos toca trabajar con esta plataforma y no teníamos ninguna idea de como usarla. Esto generó dificultades al momento de crear ramas, fusionar cambios y resolver conflictos entre archivos. El desarrollo del código no fue muy complicado, fue más difícil adaptarse al formato, aprender qué son los branches, que es un pull request, entre otras cosas. Otro aspecto complejo fue la lectura correcta de la base de

datos. Al inicio interpretamos mal el orden de las columnas del archivo CSV, lo que llevó a errores en la carga de datos. Además, olvidamos eliminar la primera fila con los títulos, lo que también provocó fallos durante la ejecución del programa. Estos problemas nos obligaron a revisar y ajustar cuidadosamente la función de lectura.

Lo más interesante de la tarea fue entender cómo estructurar un programa real en C de forma modular, trabajando con varios archivos y funciones interconectadas.

Al enfrentar errores o bugs durante el desarrollo, primero analizamos el comportamiento del programa para identificar qué parte del código estaba generando el problema. Una vez detectado el origen del error, recurrimos a herramientas como Chat GPT para explicar lo que ocurría y buscar posibles soluciones. Nuestro sistema de pruebas consistió en desarrollar cada función de forma independiente, verificando que cumpliera correctamente su propósito antes de integrarla al resto del programa. Para validar los resultados, realizamos cálculos manuales previos, de modo que al implementar cada métrica con ayuda de Chat GPT, pudiéramos comparar los resultados y localizar fácilmente cualquier inconsistencia.

Durante el desarrollo del proyecto, aprendimos varias lecciones clave sobre la implementación en C. Al trabajar con archivos CSV, comprendimos la importancia de manejar correctamente cadenas de texto y separadores, y de validar los datos antes de procesarlos para evitar errores en los cálculos. También aprendimos a organizar el código de manera clara. Este trabajo nos permitió aplicar conceptos clave del lenguaje C en un contexto práctico y más cercano al desarrollo de software real.

El uso de la IA fue una herramienta que facilitó mucho nuestro trabajo y nos permitió hacerlo de una manera más efectiva, optimizando el tiempo y aprendiendo activamente de nuestros errores. Durante el desarrollo del proyecto, Chat GPT fue utilizado en diferentes puntos. Propuso el uso de arrays y estructuras para almacenar las métricas, permitiendo que se facilite el almacenamiento de los datos. También nos ayudó a identificar y corregir errores de sintaxis y lógica, como el uso adecuado de punteros y la validación de entradas. Las sugerencias fueron validadas mediante pruebas manuales de los cálculos y la verificación de los resultados. A medida que intentabamos correr el código y este presentaba errores, le solicitabamos a Chat GPT que nos explique cuál fue el problema y que nos ayude a solucionarlo. O incluso desde antes de iniciar a realizar el código, Chat GPT fue una gran ayuda para familiarizarse con el uso de GIT y de GitHub, como también para poder compilar C en VSCode.

Además, utilizamos GitHub Copilot para comentar el código, lo que mejoró la claridad y documentación del proyecto. Cuando descubríamos errores o problemas, analizamos la situación, y luego presentamos el problema a Chat GPT para restringir la búsqueda del error, lo que optimizó nuestro proceso de depuración y resolución de problemas.

5. Referencias

- Guía de C sobre punteros: https://www.learn-c.org/
- Aprender a usar GitHub
  https://www.freecodecamp.org/espanol/news/guia-para-principiantes-de-git-y-github/