Informe de diseño

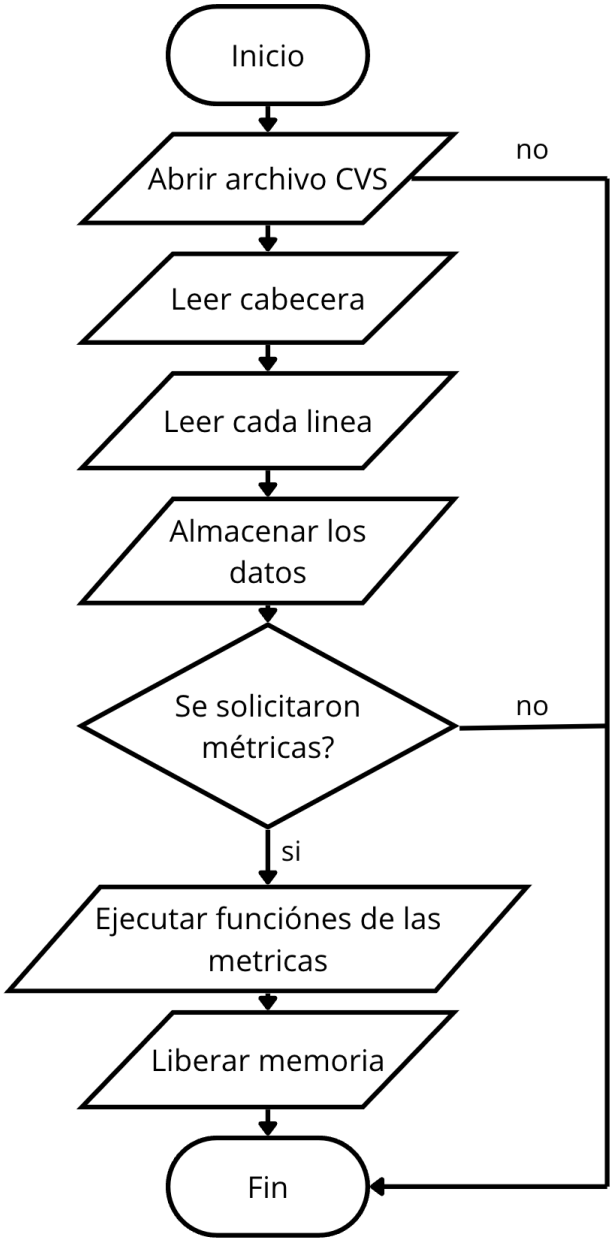
El código está estructurado en tres archivos principales: *main.c*, *utils.c* y *metrics.c*. Este enfoque modular facilita la organización del código, separando en distintos archivos las distintas funciones que el programa utilizara. En *main.c* se encuentra el flujo principal del programa, se ingresan las peticiones del usuario y se llaman las funciones necesarias para calcular las métricas pedidas. Por otro lado, *utils.c* contiene las funciones auxiliares para el procesamiento de los datos *trim()* y *parse_csv_line()*, es decir, procesa el archivo CVS para luego poder calcular las métricas. Mientras que, *metrics.c* contiene las funciones encargadas de calcular las métricas solicitadas por el usuario. Esta división permite un código más ordenado y fácil de mantener, ya que cada archivo tiene una responsabilidad clara.

Cada métrica es implementada como una función independiente dentro de *metrics.c*. Estas funciones toman como entrada el arreglo de órdenes y retornan un el resultado correspondiente que se le muestra al usuario más adelante. Este diseño modular permite agregar nuevas métricas y editar y mejorar cada función sin modificar el programa en su totalidad, minimizando errores que pueden ocurrir tras modificar el código.

El uso de punteros a funciones es importante en este diseño. En *metrics.c*, se emplea un arreglo de punteros a funciones para almacenar las direcciones de las funciones para calcular las métricas. Esto se logra con *MetricFunc*, un tipo de puntero que apunta a las funciones asociadas a cada métrica. Este enfoque modular permite que el código sea más flexible, ya que cada función de métrica se puede ejecutar de forma dinámica según sea necesario. Además, el uso de punteros a funciones optimiza el rendimiento al evitar el uso de estructuras condicionales, lo que mejora la eficiencia en la ejecución del programa.

Para leer y procesar el archivo CVS, se utilizó un método de parseo en *utils.c*. La función parse_cvs_line() recorre cada línea del archivo, separándola en tokens que son procesados y almacenados en la estructura *Order*. Este enfoque es flexible, ya que permite manejar un número variable de columnas y variables. Además, este método es particularmente útil para manejar ingredientes y nombres de pizzas, que pueden tener una cantidad variable de tokens.

El siguiente diagrama de flujo muestra de manera general el flujo del programa:



Reflexiones finales

Lo más complejo y desafiante en la tarea fue lograr desarrollar un parser robusto para el archivo CSV en C, específicamente al tener que lidiar con campos que incluían comas y comillas, los cual, si no estaban correctamente ubicados, al compilar y ejecutar la aplicación, las funciones tomaban datos incorrectos; por ejemplo, al pedir la pizza más vendida (pms), en vez de devolver el nombre de la pizza, entregaba los ingredientes, lo que nos complicó bastante, pero logramos resolverlo. Fue interesante apreciar cómo los pequeños detalles, como la forma en la que se deben reconstruir los campos o la cantidad de tokens, llegaban a afectar todo el procesamiento de datos. La implementación modular de las funciones métrica nos ayudó a poder facilitar las futuras modificaciones o amplificaciones, pudiendo mantener un código organizado.

Además del reto del parser, enfrentamos varios problemas durante el debugging, especialmente relacionados con la asignación dinámica de memoria y el maneja de cadenas en C, lo que nos llevó a tener que profundizar en conceptos que antes eran teóricos. Para poder resolver estas situaciones, usamos distintas estrategias: revisamos parte por parte el código, probando con distintos casos de prueba, para finalmente utilizar IA, como ChatGPT para que nos diera sugerencias e ideas de como estructurar el código y detectar errores, pero siempre revisando que las soluciones propuestas fueran correctas. Con esa tarea, logramos aprender la importancia que hay en trabajar colaborativamente, gracias a GIT, con lo cual pudimos ir revisando los avances de cada uno, además de revisar cuidadosamente cada parte del código para tener una solución correcta y efectiva.

Explicación de IA

Durante la escritura del código utilizamos la herramienta de inteligencia artificial ChatGPT como ayuda para resolver problemas de la estructura y depuración del código. En particular, le preguntamos cómo organizar las funciones de cálculo de métricas y cómo trabajar de mejor manera con memoria dinámica en C. También, a medida que íbamos probando el código nos ayudó para identificar errores en el código; en general, era problemas de sintaxis o relacionados a GitHub. En algunos casos, ChatGPT sugirió alternativas para mejorar la eficiencia del programa, como el uso de punteros a funciones para evitar estructuras condicionales. Había veces que ChatGPT recomendaba funciones nuevas, que no conocíamos, las cuales revisábamos su definición y uso en https://learn.microsoft.com/. Nos preocupamos de entender el código recomendando antes de incorporarlo en nuestro programa.