Informe de diseño y justificación de la solucion

El objetivo de este informe es explicar la estructura del programa desarrollado, describir las elecciones de diseño y las estructuras de datos empleadas, así como justificar la modularidad del código y el uso de punteros a funciones en el proyecto de análisis de ventas de pizzas.

El proyecto se organiza en varios archivos fuente que contienen diferentes responsabilidades:

1. **main.c**: Este archivo contiene la función principal, la cual es responsable de leer los parámetros de entrada y controlar el flujo del programa. A través de este archivo se procesan los comandos del usuario (como pms, pls, etc.) y se llaman las funciones correspondientes para calcular las métricas.
2. **parser.c**: Aquí se implementa la función que lee el archivo CSV y parsea los datos en la estructura struct order. Esta estructura es fundamental para almacenar los datos de cada orden de pizza, como el nombre, la cantidad, el precio, la fecha, entre otros. Es un componente clave que facilita el procesamiento y el análisis posterior.
3. **metrics.c**: Contiene las funciones que calculan las métricas solicitadas, como la pizza más vendida, la fecha con más ventas, etc. Cada función en este archivo es modular y está dedicada a un tipo específico de análisis.
4. **utils.c**: Este archivo incluye funciones auxiliares que permiten simplificar la implementación de las métricas y mejorar la organización del código. Por ejemplo, aquí se encuentran funciones para la limpieza de memoria o para realizar tareas repetitivas en otras partes del programa.

Estructuras de Datos

La estructura principal utilizada en este proyecto es `struct order`, que almacena los detalles de cada orden de pizza, tales como:

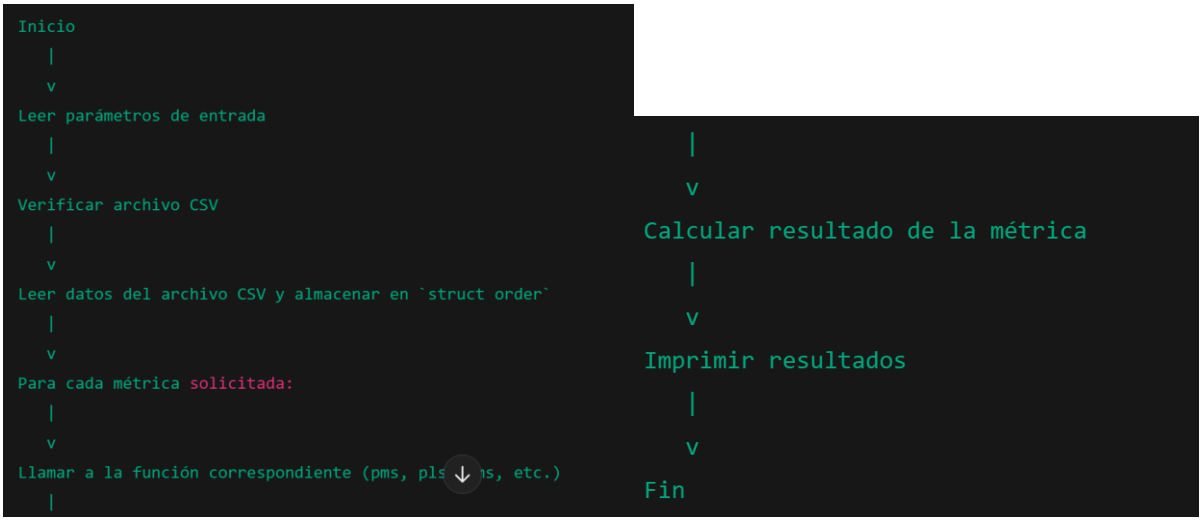
```
struct order {
    int pizza_id;
    int order_id;
    char pizza_name_id[50];
    int quantity;
    char order_date[20];
    char order_time[20];
    float unit_price;
    float total_price;
    char pizza_size[5];
    char pizza_category[50];
    char pizza_ingredients[200];
    char pizza_name[100];
};
```

Justificación de la Modularidad y Uso de Punteros a Funciones

La modularidad en el código se logra dividiendo el proyecto en múltiples archivos y funciones pequeñas, cada una con una responsabilidad bien definida. Esto hace que el código sea más fácil de mantener y depurar, además de permitir la reutilización de funciones.

- **Modularidad**: Las funciones que calculan las métricas (`pms`, `pls`, `dms`, etc.) están implementadas por separado, lo que facilita su modificación o ampliación en el futuro. Por ejemplo, si se desea añadir una nueva métrica o cambiar la forma en que se calcula alguna de las existentes, solo es necesario modificar la función correspondiente.
- **Uso de punteros a funciones**: Se utilizaron punteros a funciones para manejar dinámicamente las métricas que el usuario desea calcular. Esto permite que el programa sea más flexible y se pueda extender fácilmente agregando nuevas métricas sin tener que modificar el flujo principal del programa. Esta técnica también facilita la lectura del código, ya que las métricas se pueden invocar mediante una sola llamada a la función `get_metric`.

Diagrama de Flujo General: Este flujo refleja cómo el programa procesa los datos y las métricas de forma secuencial.



Razones de Diseño

Parseo del CSV: Se eligió utilizar la librería estándar de C para leer el archivo CSV línea por línea y luego parsear los datos utilizando `strtok` para separar los campos. Esta técnica es eficiente y directa para leer datos en formato CSV y permite manejar fácilmente los campos separados por comas.

Almacenamiento de Ingredientes: Los ingredientes se almacenan como cadenas de texto separadas por comas en la estructura `pizza_ingredients`. Esto facilita la búsqueda y el conteo de ingredientes, ya que solo es necesario buscar las subcadenas dentro de cada string para contar las veces que un ingrediente aparece.

Interacción entre Archivos

- **main.c** gestiona la entrada del usuario, invoca las funciones de cálculo de métricas y maneja la salida.
- **parser.c** se encarga de leer el archivo CSV y devolver un arreglo de `struct order` con los datos.
- **metrics.c** contiene las funciones que calculan las métricas y devuelven los resultados formateados.
- **utils.c** incluye funciones de utilidad para tareas auxiliares, como la limpieza de memoria o el manejo de cadenas.

Reflexiones Finales y Autoevaluación

Lo más complejo o interesante: La parte más interesante fue diseñar un sistema flexible y modular que pudiera manejar múltiples métricas sin hacer el código excesivamente complejo. El uso de punteros a funciones fue clave para lograr esto, ya que permitió que el programa fuera fácilmente extensible.

Cómo enfrentaron los errores, pruebas y debugging: Los errores se enfrentaron utilizando un enfoque iterativo de prueba y depuración. Al principio, se escribieron pruebas simples para verificar la entrada de datos y el correcto parseo del CSV. Luego, se verificó que cada métrica produjera resultados correctos. Durante el proceso de debugging, se utilizaron herramientas como `printf` para rastrear los valores de las variables y verificar la lógica de las funciones.

Lecciones aprendidas: Implementar esta solución en C nos permitió comprender mejor el manejo de memoria y las técnicas de optimización de recursos, como el uso de punteros y la liberación de memoria. Además, aprendimos cómo organizar y estructurar un proyecto de programación de manera modular y eficiente.

Uso de IA: Durante el desarrollo, ChatGPT fue utilizado para sugerir mejoras en las funciones, especialmente en lo que respecta a la manipulación de cadenas y el uso eficiente de memoria. También se usó Copilot para revisar y corregir errores en las funciones que calculan las métricas, era nuestra mayor fuente de ayuda ya que se manejaba como coworker en github y sabia toda la estructura y archivos que creabamos en el repositorio, ayudandonos en la busqueda de los problemas que tuvimos al conectar varios archivos para que las funciones del programa funcionaran. La información proporcionada por la IA fue validada a través de pruebas y depuración para asegurar que los resultados fueran correctos.

Conclusión

El proyecto fue un desafío que permitió mejorar nuestras habilidades en programación en C, especialmente en el manejo de archivos, estructuras de datos y optimización de recursos. La solución modular y el uso de punteros a funciones han hecho que el código sea flexible y fácil de mantener, lo que facilitará la ampliación futura del programa.

Inte