UNIVERSIDAD ADOLFO IBÁÑEZ

Informe de diseño, Aplicación 1.

Nombres: Felipe Retamal, Alonso Paniate.
Profesor: Justo Vargas
Lenguajes y paradigmas de la programación, sección 2.

Introducción:

El proyecto de la primera Aplicación (APP 1) consiste en una aplicación en lenguaje C para el análisis de ventas de una pizzería. La aplicación tiene como misión leer un archivo CSV con información de ventas y calcular distintas métricas (por ejemplo, pizza más/menos vendida, fecha con mayor recaudación, promedio de pizzas por orden, etc.) utilizando un diseño modular y el uso de punteros a funciones.

Informe de Diseño y Justificación de la Solución:

El proyecto se ha organizado en varios archivos fuente, lo que permite separar las responsabilidades y facilitar el mantenimiento:

- **main.c**
  Contiene la función principal "Main" que procesa los argumentos de línea de comandos, carga los datos del archivo CSV y mapea cada métrica solicitada a su función correspondiente mediante punteros a funciones.

- **order.h**
  Define la estructura "Order", la cual representa cada registro de venta del CSV. Esta estructura incluye campos como identificadores, fecha, hora, cantidad, precios, ingredientes y demás datos relevantes para cada venta.

- **utils.c / utils.h**
  Se encargan del parseo y la lectura del archivo CSV. La función "load_orders" abre el archivo, ignora la cabecera, lee línea a línea y convierte los datos a los tipos correspondientes, almacenando estos en un arreglo dinámico de estructuras llamado "Order".

- **metrics.c / metrics.h**
  Implementan las funciones para el cálculo de cada métrica solicitada. Cada función tiene la firma:

  ```
  // Declaración de las funciones de métricas
  char* metric_pms(int *size, Order *orders);
  ```

  y utiliza punteros a funciones para permitir que el programa principal invoque la función correspondiente según el parámetro ingresado. Se incluyen métricas como pizza más vendida (pms), pizza menos vendida (pls), fecha con más o menos ventas (dinero o cantidad de pizzas), promedios y agrupación por categorías o ingredientes.

- **Makefile**
  Automatiza la compilación del proyecto, definiendo las dependencias y reglas para generar el ejecutable "app 1".

Modularidad y Separación de Responsabilidades

La modularidad en nuestro proyecto se logra al dividir el código en archivos .c y .h, lo que permite separar claramente la implementación de las funcionalidades de sus interfaces. Los archivos “.c” contienen la lógica de programación, es decir, el código que ejecuta las funciones y procesa los datos, mientras que los archivos .h son en los cuales se declaran las estructuras, constantes y prototipos de funciones que serán utilizadas por otros módulos del programa. Por ejemplo, en nuestro caso, se creó el archivo order.h para definir la estructura Order, la cual contiene todos los campos necesarios para representar una venta, permitiendo que cualquier módulo pueda acceder a esta información sin conocer los detalles internos de su implementación. De manera similar, los archivos utils.h y metrics.h declaran los prototipos de las funciones encargadas del parseo del CSV y del cálculo de métricas, respectivamente, facilitando la comunicación entre módulos como main.c, utils.c y metrics.c. Esta organización no solo mejora la claridad y mantenibilidad del código, sino que también permite una compilación incremental y la escalabilidad del sistema, ya que la incorporación o modificación de nuevas funcionalidades se puede realizar de forma aislada sin afectar al resto del proyecto.

Además, se ha implementado un arreglo de punteros a funciones en “main.c” que permite mapear cada nombre de métrica (por ejemplo, "pms" o "dls") a la función que la implementa. Esta implementación basada en punteros a funciones hace que la solución sea más flexible y escalable, ya que agregar o modificar métricas se reduce a actualizar el mapeo, sin necesidad de alterar la estructura central del programa.

Métodos de Parseo del CSV

En nuestro proyecto, el método de parseo del archivo CSV se implementó en el módulo utils.c. Para leer los datos, se utiliza la función “fgets” para obtener cada línea completa del archivo, lo que permite procesar el contenido línea por línea, comenzando por omitir la cabecera. Una vez leída una línea, se emplea strtok para dividirla en campos utilizando la coma como delimitador. Este enfoque, aunque básico, es adecuado para el formato esperado del CSV, ya que se asume que los datos son válidos y no contienen campos vacíos. Además, se utilizan funciones como “atof” y “atoi” para convertir las cadenas leídas a los tipos numéricos correspondientes, y se implementa una estrategia de asignación dinámica de memoria con “malloc” y “realloc” para manejar un número variable de registros. Así, la información se organiza en la estructura “Order”, permitiendo un acceso ordenado y eficiente a cada uno de los datos, lo que facilita el posterior cálculo de las métricas solicitadas.

Interacción entre archivos

En nuestro proyecto, la interacción entre archivos se establece de manera que:

- **main.c** actúa como el punto de entrada y coordinador del programa, procesando los argumentos de la línea de comandos y gestionando la ejecución de las funciones.

- **utils.c** se encarga de la lectura y el parseo del archivo CSV, convirtiendo cada línea en una estructura Order.

- **metrics.c** contiene las funciones encargadas del cálculo de las métricas, las cuales son invocadas desde main.c mediante un arreglo de punteros a funciones.

Esta organización permite que cada módulo se concentre en una tarea específica, facilitando tanto la implementación como el mantenimiento del sistema. En cuanto a las referencias de recursos externos, se consultaron las siguientes fuentes durante el desarrollo:

- Documentación oficial del lenguaje C.

- Foros y comunidades de programación para resolver dudas específicas sobre manejo de archivos y memoria dinámica.

- Uso de IA como Chat GPT o BlackBox AI para entender y resolver errores de ejecución y/ o compilación.

Principales desafíos encontrados

El proyecto presentó diversas complicaciones principalmente debido a la inexperiencia en C y ubuntu, así como también en problemas de lectura o formato con el csv usado. Una de las principales dificultades fue la correcta lectura del archivo CSV, ya que los formatos de datos no se leian correctamente y por ejemplo al pedir por la métrica pms (Pizza más vendida) los códigos iniciales arrojaban solamente la cantidad vendida, sin especificar en cuál era la pizza de la que se trataba o la cantidad, lo que llevaba a errores en la interpretación de la información. Además, se incurrió en muchos errores de tipeo o lógica de programación del lenguaje debido a la inexperiencia de programación en C. Además la misma inexperiencia en el entorno de Ubuntu también presentó sus propios retos; la necesidad de utilizar la terminal para compilar y ejecutar el código resultó complejo e hizo que tomar apuntes de los comandos fuera una de las principales tareas, por ejemplo, previo a tener el “makefile” se cometieron muchos errores a la hora de ejecutar o compilar los archivos, lo que hacía mucho más lento y desorganizado el desarrollo por que muchas veces no sabíamos porque había fallado el código, solo para descubrir que había sido un error con la compilación de los archivos o la ejecución. Además, el parseo de datos desde el archivo CSV implicó dividir cadenas y convertir tipos de datos, lo que se complicó por la necesidad de manejar

correctamente los separadores y los espacios en blanco, ya que el parseo incorrecto llevaba a datos y respuesta erróneas que dañaron el resultado final del código. Por otro lado, entender e integrar la programación estructurada, paradigmas procedural y funcional, también generó confusión y dificultades a la hora de estructurar y desarrollar el código. Finalmente, separar las funciones, mantener la modularidad y asegurarse de que las funciones se comunicaran correctamente, complicó mucho el correcto desarrollo del proceso.

Uso de IA:

El uso de IA en el trabajo realizado fue fundamental, ya que ayudó a simplificar nuestro trabajo, siendo usado principalmente por nosotros en tres puntos, en la creación de la estructura del programa, en el parseo del CSV y en el debugging.

Principalmente fueron utilizadas Blackbox.IA y Chat GPT, donde especialmente en el área de debugging y explicación de los errores para así poder solucionarlos fue donde mayor ayuda ofrecieron estas.

La IA nos ayudó a generar el flujo principal del programa, permitiéndonos definir claramente las responsabilidades de cada módulo, se establece que “main.c” se encargará de leer los argumentos de la línea de comandos de la terminal, validar la entrada y coordinar la ejecución de las métricas mediante punteros a funciones.

Respecto al parseo del CSV, está nos propuso elegir “fgets” para obtener cada línea completa del archivo o también por ejemplo “strtok” para dividirla en campos utilizando la coma como delimitador, lo cual fue fundamental para realizar de forma correcta el parseo del CSV.

Durante el proceso de debugging, la IA facilitó la identificación de errores y advertencias, entregando explicaciones claras de porque ocurrían estas y soluciones prácticas para resolverlos. Además, nos ayudó a generar archivos CSV de prueba con datos en el mismo estilo del ejemplo, lo cual nos permitió validar el correcto funcionamiento de todas las métricas. La creación de estos CSV de prueba fue fundamental para verificar que los cálculos (por ejemplo, la suma de cantidades, promedios y agrupaciones) se realizaban de acuerdo con lo esperado, y para ajustar el código cuando aparecían diferencias.

Reflexión final

El desarrollo de este proyecto sirvió para el aprendizaje activo, enfrentándonos a complicaciones reales al usar C, Ubuntu o al parsear archivos CSV. Esto nos permitió mejorar y entrenar habilidades críticas para la resolución de problemas y la programación, así como comprender los desafíos que pueden surgir al programar, cómo resolverlos, corregirlos o incluso mejorar y optimizar nuestra forma de programar, algo esencial en este campo. Finalmente, para el correcto desarrollo del proyecto, fue fundamental aprender a mantener

una organización y un orden adecuados al programar, lo que facilitó la colaboración dentro del equipo. Además, fue clave comprender la lógica detrás de las estructuras de datos utilizadas y cómo se aplica a la realidad la teoría vista en clases sobre paradigmas de programación, como la programación funcional, procedural y estructurada.