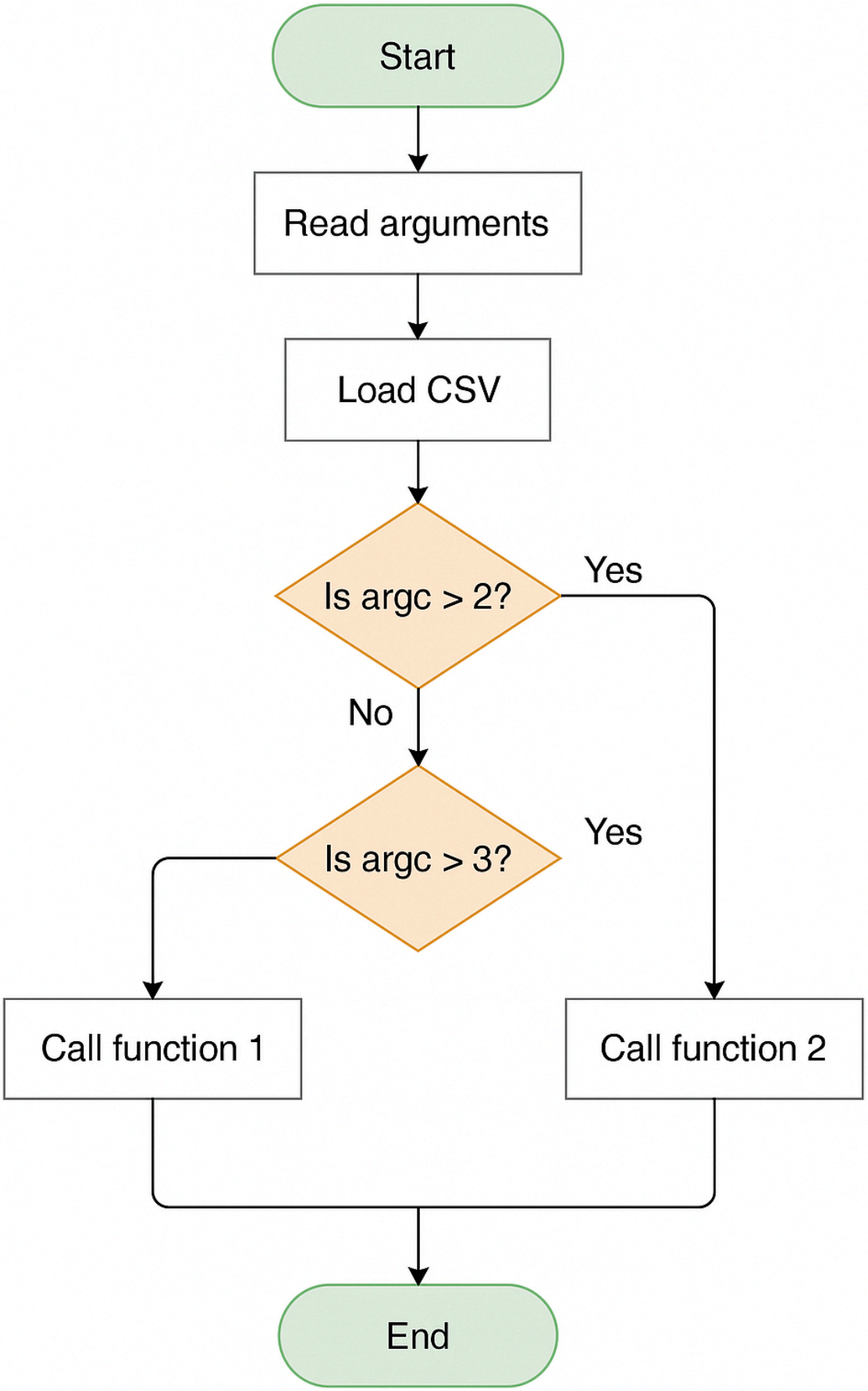
Informe de diseño y justificación de la solución

- **Objetivo:** El código tiene una estructura modular que se divide en main.c como el flujo principal, la carpeta de src que contiene el único archivo pizza_analysis.c como función de análisis y la carpeta includes que contiene el único archivo pizza_analysis.h con todas las declaraciones necesarias, junto con el archivo de Makefile que llama todos los archivos anteriores para compilar el código.
  Asimismo, dentro de la estructura del código como tal, es importante hablar de PizzaOrder como un struct clave para organizar el tipo de datos, el Order ha servido para almacenar lo pedido exitosamente.

- **Razones de diseño:** El parseo del csv paso por distintas etapas, en primer lugar se ocupo sscanf para leer el csv y adjudicar a su variable correspondiente de una sola vez, sin embargo, tuvo algunos problemas a la hora de almacenar order_date, debido a esto consultamos al ayudante y nos propuso ocupar strtok, trabajamos con la nueva función y aunque el código de lectura es mucho más largo de esta manera (definir uno a uno hasta que punto leer) nos aseguramos que todo fuera recibido de manera adecuada.

- **Explicación de la interacción entre archivos h. y c.**
  Los archivos .h y .c son relevantes para la organización modular, asimismo, el .h se usa como una interfaz de un encabezado y el .c hace la implementación de aquellos encabezados declarados.

  Seguido de esa misma lógica, el archivo .c de un módulo siempre incluye su propio .h tal como es en nuestro programa ya que pizza_analysis.c tiene sí o sí su archivo pizza_analysis.h. Esto hace que el compilador verifique que la implementación en el .c coincida con la interfaz declarada en el .h. Esto da la información necesaria para llamar a las funciones y usar las estructuras de datos.
  La interacción entre h. y c. depende del compilador, sin embargo, mientras en el archivo .c existan las funciones que se declararon en h., podrá entender el código y funcionar sin problema. El enlazador luego combina el código compilado del archivo .c para crear el programa final y funcionar como si fuera uniforme.
- **Diagrama de flujo**


Start
Read arguments
Load CSV
Is argc > 2?
Yes
No
Is argc > 3?
Yes
Call function 1
Call function 2
End

Sección de reflexiones finales o autoevaluación

Reflexión del Ivan:

¿Qué fue lo más complejo o interesante de la tarea?

Lo más complejo para mí fue configurar la correcta lectura y guardado del csv, también era interesante ver cómo la struct tomaba forma y era super útil a la hora de guardar las variables como una mini base de datos.

¿Cómo enfrentaron los errores, pruebas y debugging?

Cómo la mayoría de problemas estaban relacionados con la lectura del csv incluimos varios printf en el desarrollo para ver hasta qué línea o que columna se leía correctamente, también creamos un csv con formato idéntico al indicado pero más grande para probar con mayores datos las métricas. Cuando pedimos código a las IA (Gemini y Chat GPT) usualmente no venía del todo funcional, así que también tuvimos que corregir esas líneas un poco.

¿Qué lecciones aprendieron al implementar en C este tipo de lectura de archivos y cálculos de métricas?

Aprendí mucho sobre especificadores de formato luchando por que sscanf impartiera bien los tipos de variables, la importancia que tiene tener un formato estándar de csv porque sin un formato consistente esto hubiese sido 10 veces peor, por lo que el preprocesamiento de datos siempre es relevante. Al hacer las métricas facilita mucho el trabajo tener una struct bien organizada de donde obtener las variables a ocupar.

Reflexión de Camila:

¿Qué fue lo más complejo o interesante de la tarea?

Personalmente, como me dediqué al 100% en que el Makefile funcionará, para mí fue más complejo el poder hacer que todo funcionará y se enlazará bien, estructurar una tarea así en partes fue una experiencia nueva para mí y aunque entendía conceptos, sufría constantemente porque el Makefile o las rutas de mis carpetas con distintos archivos no se encontraba, osea tenía errores de conexión con las carpetas y etc, aún así, fue interesante de aprender y creo que estoy preparada para volver a tener que lidiar con Make.

¿Cómo enfrentaron los errores, pruebas y debugging?

Me di cuenta por mí parte, que mis errores se debían por cosas muy simples que simplemente se me pasaban de largo, al inicio tenía errores prácticos de tabulador en el código del Makefile y yo me frustraba de no saber qué hacer. Luego de por sí, errores de código no tuve, pero tuve un error particular que se debía a la ruta de mis includes, me guíe para solucionar este error con mucha ayuda de la ia ya que ciertamente empecé a perderme. Algo que también me di cuenta, es que yo como usuaria de Linux, había trabajado en el código de Makefile con instrucciones de codificación específica para Linux, por lo tanto, tuve que corregir ese pequeño error al momento de querer ejecutar el programa como prueba en computadores distintos con sistemas Windows o etc, así, todo sobre app1 pudo funcionar en cualquier dispositivo de mis compañeros de grupo luego de ese pequeño arreglo en los comandos.

¿Qué lecciones aprendieron al implementar en C este tipo de lectura de archivos y cálculos de métricas?

Honestamente, yo creo que aprendí bastante bien lo que es la estructuración de un código al aplicar esa lógica modular, me ha quedado el concepto clave de ello y creo que hasta en el futuro podría emplear este mismo tipo de dinámica de organización ya que yo lo encontré práctico, puede que con otros lenguajes sea distinto pero me ha gustado y he aprendido cosas algo distintas para mí como lo fueron modificar las variables de entorno y acceder a las rutas de los archivos con los que se trabaja.

Reflexión de Emilio:

¿Qué fue lo más complejo o interesante de la tarea?

Lo más interesante desde mi punto de vista fue la experiencia y el aprendizaje al realizar el código, si bien es cierto que tenemos conocimientos básicos de programación de python, eso no nos asegura que nos manejemos bien en otro lenguaje como lo es c, por lo que a la hora de realizar el código fue una experiencia en la que nos desafiamos a nosotros mismos y nos da la posibilidad de aprender de forma autodidacta, fomentando el uso de nuestras capacidades y cómo nos desenvolvemos en situaciones laborales. Con respecto a la realización del código, considero que lo más complejo fue entender los directorios de visual studio, junto con github, la instalación de gcc y su uso para compilar el código. Me resultó bastante complejo ya que desde python siempre trabajamos desde un mismo directorio, y simplemente teníamos que descargar las librerías si era necesario. Este ejemplo fue lo que más me provocó estrés y frustración de mi parte. Lo interesante y entretenido fue a la hora de organización y realización del código, después de poder solucionar el problema viene la parte de codificación, y la ansia del print que nos entregaría nuestro código, ¿estará bien?, ¿se habrá arreglado?, ¿cómo puedo solucionarlo? fueron las preguntas que pasaban por mi cabeza y hacían que mi percepción del tiempo fluyera.

¿Cómo enfrentaron los errores, pruebas y debugging?

Muchos de los errores se solucionaron viendo tutoriales de youtube, con inteligencia artificial, y consultando a compañeros. El mayor problema que tuve, como explique antes, fue con los directorios, el gcc, la terminal de windows, los cuales no fueron problemas de código en sí. Para esto utilice mucha inteligencia artificial, preguntando cómo funcionaban, como cambiaba de directorio, como compilar el código desde el cmd de windows, además acudí a la ayuda de mis compañeros quienes me ayudaron a comprender el problema que tenía. Muchas veces me pasó también que intentaba correr el código pero no tenía compilado las modificaciones que le hice al archivo, por lo que no cambiaba el print del terminal.

¿Qué lecciones aprendieron al implementar en C este tipo de lectura de archivos y cálculos de métricas?

Aprendí bastante sobre la estructuración de los códigos, es importante tener un orden a la hora de hacer un código ya que es más fácil identificar el problema, además puedes ir trabajando en partes lo que facilita mucho más que partes funcionan y cuáles no. Considero que la parte más importante es como uno carga el archivo en el programa y como uno quiere trabajar con este, ya que si el archivo no se carga bien es muy difícil obtener resultados correctos con respecto a las métricas.

Explicación de cómo se usó IA en nuestro Proyecto:

Las inteligencias artificiales fueron Gemini (Google) y ChatGPT (OpenAI)

Es importante recalcar que el uso de la IA facilitó mucho la confección del código, ya que nos ahorró tiempo en problemas que quizás no eran tan importantes. Como ejemplo muchas veces faltaron cerrar paréntesis o finalizar alguna parte del código con un punto y coma, por lo que la inteligencia artificial nos ayudó mucho a identificar estos pequeños problemas para que nuestro código pudiera correr.

La inteligencia artificial nos ayudó mucho a entender cosas que nosotros no sabíamos, como por ejemplo, a la hora de pedir en consola el ejecutable del archivo, el archivo csv y las métricas. Esto era algo que nosotros desconocemos en su totalidad por lo que la IA nos ayudó a entender cómo funcionaba este formato y cómo aplicarlo. Se pidió ayuda también a la inteligencia artificial porque teníamos errores al utilizar la función sscanf donde solo reconocía y leía 4 columnas, pero en general teníamos errores con esta función. Así que después de algunas consultas con el ayudante, llegamos a la conclusión que sería mejor utilizar strtok, por lo que le pedimos a la inteligencia artificial que nos ayudará a estructurar esta parte del código. Con la elaboración de las funciones no tuvimos mayor problema, solo con algunas que pedimos ayuda a la IA, ya que por ejemplo, con la función dms, en un

inicio solamente nos entregaba el valor de la pizza con mayor precio de ese dia, pero no la suma del total de las ventas del dia, asi que con un poco de ayuda con la IA pudimos reestructurar agregando algunas cosas a nuestro código.

El código asimismo, al haberlo hecho de una forma uniforme desde el inicio no tenía lo más importante que era la forma estructura modularmente. Por lo mismo, gracias a la IA se pudo ordenar cómo correspondía tanto los archivos c. como h. junto con el main.c y el archivo Makefile, así, pudimos estructurar el código de la forma correcta en las carpetas correspondientes y hacer que el make funcionará.