UNIVERSIDAD ADOLFO IBÁÑEZ

Informe de diseño grupo 6

Profesor: Justo Miguel Vargas.

Ayudante: Diego Eduardo Duhalde Venegas

Grupo 6

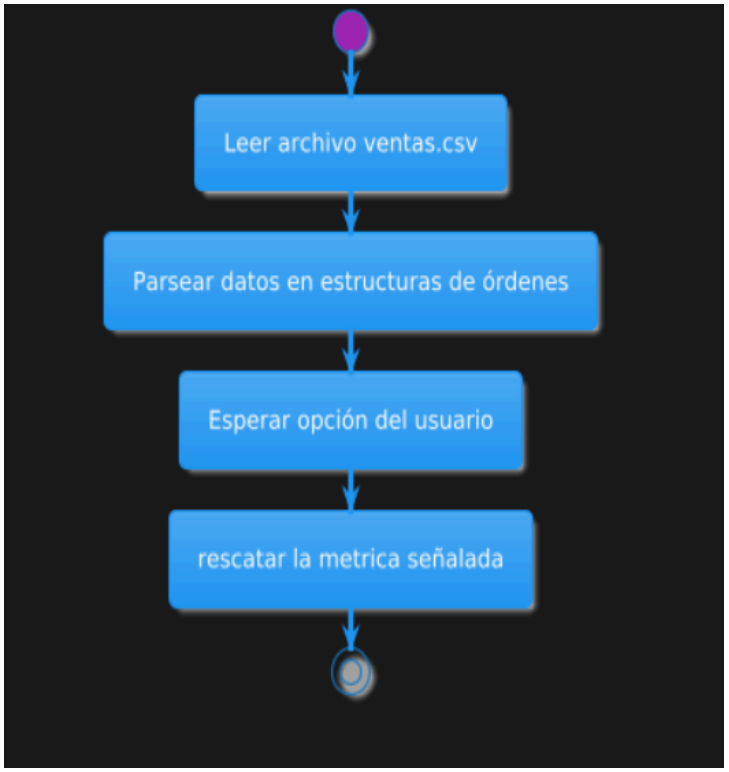
Ignacio Vargas Latorre, Vicente Blanco Farías, Flavio Bustamante Farah

1. Arquitectura:

La arquitectura general de nuestro proyecto “App1” fue diseñada en base al prompt recomendado por el profesor en clase (ver prompt en x.x), con este prompt Chat GPT nos sugirió un diseño basado en arquitectura modular, organizada en archivos fuente (.c) y de cabecera (.h). El archivo main.c es la función principal del programa, esta se encarga de coordinar y llamar las diversas funciones y métodos del programa, sus funcionalidades son las siguientes:

1. csv_reader.h y csv_reader.c: Estos archivos son los que se preocupan de la lectura y el parseo correcto del .csv que el usuario le otorgue al programa.
2. order.c y order.h: Estos archivos definen la estructura de datos que representa los pedidos del archivo csv, este proporcionado por “csv_reader.c”, entregando de vuelta funciones para almacenarlos, buscarlos y acceder a sus atributos.
3. metrics.c y metrics.h: Estos archivos son los que contienen las diversas métricas que se pueden solicitar en el programa, como lo son “la pizza más vendida”, “pizza menos vendida”, etc. Estas métricas operan sobre los datos leídos en “csv_reader.c”.

Aquí tenemos un diagrama de flujo que explica en muy simple forma el funcionamiento general del código:



2. Uso de punteros:

Uno de los aspectos clave del App1 es la implementación del uso de punteros a funciones para aplicar métricas de manera flexible. Cada métrica está implementada como una función con una firma común, lo que permite almacenar referencias a estas funciones en un arreglo de punteros. Gracias a esto podemos utilizar la misma información para cada una de las funciones y nos permite obtener múltiples con solo una llamada.

En nuestro caso tenemos que la firma común de las funciones que utilizamos es:

char* “metrica deseada”(int *size, Order *orders)

Esto permite que a partir de una misma información proporcionada se puedan aplicar diversas funciones.

3. Lectura del CSV:

La lectura del .csv fue la parte más compleja de todo el desarrollo de la app1. En un principio pensamos en utilizar la función strtok() para así leer el csv en el archivo “csv_reader.c”, pero rápidamente nos encontramos con un problema a la hora de utilizar esta función, ya que con ella los parámetros que recibe son el string como tal y el separador, con este último fue que tuvimos el problema. En el .csv, el apartado de ingredientes tiene comillas al inicio y al fin para así poder enumerar múltiples ingredientes y que aún así mantenga todos los ingredientes en la misma columna, el problema yace en que no podíamos colocar las comillas como separador, por lo que la función terminaba retornando cada ingrediente como una categoría distinta. Esto causó algunos problemas importantes, como que en nuestro .csv de referencia teníamos esta línea:

```
pizza_id,order_id,pizza_name_id,quantity,order_date,order_time,unit_price,total_price,pizza_size,pizza_category,pizza_ingredients,pizza_name

3.00,2.00,five_cheese_l,1.00,1/1/2015,11:57:40,18.50,18.50,L,Veggie,"Mozzarella Cheese, Provolone Cheese, Smoked Gouda Cheese, Romano Cheese, Blue Cheese, Garlic",The Five Cheese Pizza
```

Entonces, según nuestra función, “pizza_name” terminaba siendo leído como el segundo ingrediente de la pizza y no el nombre real de la pizza, por lo que en todas las métricas donde requirieron el nombre de la pizza nos entregaba uno de los ingredientes.

Para solucionar nuestro problema pensamos durante mucho tiempo en formas en las que podíamos leer el .csv de tal manera que las comillas no hicieran problema, terminamos pensando en separar el trozo con los ingredientes, pero no estábamos seguros, por lo que de todas maneras decidimos consultar nuevamente con Chat GPT, este nos aconsejó hacer lo que pensábamos y utilizar un método en el que se lee linea por linea el .csv y se van eliminando los elementos fuera de las comillas, para luego guardar lo interno como un elemento por separado, de esta forma podemos previamente leer todo lo que no es ingrediente y luego tener por separado los ingredientes como un string aparte, así, la cantidad de ingredientes no perjudica la cantidad de columnas ni su posición.

4. Modularidad:

En general nuestro programa está separado en múltiples archivos para así facilitar en la modularidad:

- “csv_reader”: decidimos dejar este archivo por separado ya que de esta forma es fácil cambiar el tipo de archivo o la forma de leerlo, es cosa de crear un nuevo archivo con el nuevo método y cambiar la mención de este en el resto de archivos.
- “orders”: en caso de tener que modificar la organización de la estructura de las órdenes de la pizzería, dejamos este archivo por separado, así su modificación es fácil.
- “metrics”: para mayor organización mantuvimos todas las funciones de las métricas pedidas en un archivo por separado.

5. Autoevaluación:

Entre todos llegamos a la conclusión de que lo que fué más complejo fue el tema de leer el csv, al principio fue lo más rápido y no hubo drama, pero una vez empezamos a programar las métricas comenzamos a toparnos con muchos problemas a la hora de llevar a

cabo las métricas que tuvieran el nombre de la pizza involucrado en ellas, estuvimos mucho rato dando vueltas alrededor del problema ya que no estábamos seguros de que era lo que no estaba funcionando correctamente, pero después llegamos a la conclusión de que quizás era problema del lector de csv, y después de pensar un rato se nos ocurrió la idea de separar la parte de los ingredientes, que luego Chat GPT terminó por confirmar que era una opción viable.

Múltiples de los problemas que nos encontramos fueron solucionados conversando entre todos y discutiendo las ideas que se nos ocurrían hasta que una funcionó, los otros problemas que lograron superarnos fueron solucionados gracias a Chat GPT, aunque a veces se equivocó y tuvimos que corregirlo. Aprendimos un poco más sobre cómo funciona el tema de leer archivos, desde antes ya sabíamos cómo funcionaba gracias a cursos anteriores de la carrera, pero nunca habíamos tenido una oportunidad en el que aparecieran comillas de la forma en la que ocurrió aquí, por lo que no teníamos ni idea al principio de como tacklear esa situación. Además de eso comprendemos mejor cómo funciona el referenciar otros archivos para así poder modularizar el proyecto.

6. Transparencia y uso de IA:

En este trabajo la inteligencia artificial jugó un rol muy importante a la hora de facilitar la escritura del código y de lograr que este funcionara correctamente, con ayuda de la IA pudimos formar las bases del código en general, además de solucionar algunos problemas concretos. Verificamos el funcionamiento del código debugeando manualmente y comprobando su funcionamiento de forma directa utilizando algunos archivos .csv como ejemplo.

Aquí están los prompts que utilizamos, para el prompt del código general decidimos preguntar de dos formas distintas para poder observar qué diferencias podría hacer Chat GPT dependiendo de como se pide la generación del código:

Prompt para el código general: “Actúa como un experto en programación procedural, respetando el principio de responsabilidad única y buenas prácticas de desarrollo de software

modular. Ayúdame a crear un programa en C que lea un archivo csv que tiene este formato: "pizza_id,order_id,pizza_name_id,quantity,order_date,order_time,unit_price,total_price,pizza_size,pizza_category,pizza_ingredients,pizza_name" Asegúrate de tener todo dividido en archivos h y c según corresponda. Debo poder responder a los siguientes comandos """pms: Pizza más vendida pls: Pizza menos vendida dms: Fecha con más ventas en términos de dinero (junto a la cantidad de dinero recaudado) dls: Fecha con menos ventas en términos de dinero (junto a la cantidad de dinero recaudado) dmsp: Fecha con más ventas en términos de cantidad de pizzas (junto a la cantidad de pizzas) dlsp: Fecha con menos ventas en términos de cantidad de pizzas (junto a la cantidad de pizzas) apo: Promedio de pizzas por orden apd: Promedio de pizzas por día ims: Ingrediente más vendido hp: Cantidad de pizzas por categoría vendidas""" Que pueden venir solos o de uno en esta forma: -./app1 ventas.csv pms pls ./app1 ventas.csv pms Debes saber cómo contexto que: -El usuario siempre ingresara información valida -No habrán campos vacíos en el archivo de ventas -El orden en que se piden las métricas por pantalla es el orden en que se muestran. Debes tener en cuenta las siguientes restricciones: -debes usar punteros para cada métrica (o un arreglo de punteros a funciones), con firma; char*(métrica)(int *size, struct order¨*orders); -No se requieren validaciones extras de los datos de entrada -Se aceptan múltiples métricas en una misma ejecución -El nombre del ejecutable debe ser app1. Genera la estructura del programa, antes de pasar al Código discutamos sobre la misma y cuando te diga pasemos a la implementación ”

Segundo prompt para el codigo general: “Actúa como un experto en programación procedural, respetando el principio de responsabilidad única y buenas prácticas de desarrollo de software modular. Ayúdame a crear un programa en C que lea un archivo csv que tiene este formato: "pizza_id,order_id,pizza_name_id,quantity,order_date,order_time,unit_price,total_price,pizza_size,pizza_category,pizza_ingredients,pizza_name" Asegúrate de tener todo dividido en archivos h y c según corresponda. Debo poder responder a los siguientes comandos """pms: Pizza más vendida pls: Pizza menos vendida dms: Fecha con más ventas en términos de dinero (junto a la cantidad de dinero recaudado) dls: Fecha con menos ventas en términos de dinero (junto a la cantidad de dinero recaudado) dmsp: Fecha con más ventas en términos de cantidad de pizzas (junto a la cantidad de pizzas) dlsp: Fecha con menos ventas en términos de cantidad de pizzas (junto a la cantidad de pizzas) apo: Promedio de pizzas por orden apd: Promedio de pizzas por día ims: Ingrediente más vendido hp: Cantidad de pizzas por

categoría vendidas""" Que pueden venir solos o de uno en esta forma: -./app1 ventas.csv pms pls ./app1 ventas.csv pms Debes saber cómo contexto que: -El usuario siempre ingresara información valida -No habrán campos vacíos en el archivo de ventas -El orden en que se piden las métricas por pantalla es el orden en que se muestran. Debes tener en cuenta las siguientes restricciones: -debes usar punteros para cada métrica (o un arreglo de punteros a funciones), con firma; char*(métrica)(int *size, struct order¨*orders); -No se requieren validaciones extras de los datos de entrada -Se aceptan múltiples métricas en una misma ejecución -El nombre del ejecutable debe ser app1. Genera la estructura del programa, antes de pasar al Código discutamos sobre la misma y cuando te diga pasemos a la implementación Una nota más, te adjunto un archivo csv a modo de ejemplo, adicionalmente te adjunto un csv a modo de ejemplo para que veas a que nos enfrentamos y el resultado que deberia dar el codigo”

Corrección de la estructura: “en la estrucutra preferiria que tuviesemos la siguiente estructura; main.c, csv_reader.c, csv_reader.h, orders.h,oreders.c, metrics.c y metrics.h, ademas de ventas.csv”

Ayuda con el problema del .csv: “Estamos intentando leer un .csv con strtok, pero estamos teniendo problemas a la hora de separar los ingredientes del resto de cosas ya que no podemos especificar las comillas como delimitador, que podemos hacer para que nos separe correctamente las columnas? Estábamos pensando en hacer una función que revisara donde en el string comenzaban y terminaban las comillas y que luego guardara el contenido interno en una string separada, sería eso viable?”