Informe de diseño de software

Fecha: 05/04/2025

Felipe Álvarez Mercado

Jan Michaelsen Thiel

Justo Miguel Vargas
Lenguajes y paradigmas de la programación
Sección 2

Objetivo general

La meta principal de este proyecto fue, por un lado, desarrollar exitosamente un programa utilizando el lenguaje "C", basado en el paradigma procedural. La idea es que este programa fuera capaz de leer, procesar y analizar un archivo de ventas en formato CSV, perteneciente a una pizzería ficticia. A partir del archivo mencionado, el programa debe calcular métricas específicas mediante fórmulas matemáticas básicas, pero extrayendo los datos necesarios del archivo. Por otro lado, también se implementó la utilización de la plataforma GitHub, para así aprender el funcionamiento y el correcto uso de esta, para facilitar y agilizar el trabajo en equipo.

Organización del código

El código se dividió en múltiples archivos para mantener una estructura modular, clara y mantenible. Dicha estructura es la siguiente:

- **main.c**: contiene la función principal. Se encarga de orquestar la ejecución del programa, leer los argumentos ingresados por el usuario, llamar las métricas solicitadas y presentar los resultados por pantalla.
- **utils.c**: incluye las funciones auxiliares, es decir, el resto de funciones que no son las de métricas. Entre estas se encuentra la función que lee y parsea el archivo CSV, posteriormente cargando sus datos en la memoria. También contiene la función para liberar memoria al finalizar.
- **metrics.c**: contiene las definiciones de todas las funciones de las métricas solicitadas por el enunciado. Cada métrica se implementa como una función con la misma firma, lo que permite tratarlas de forma genérica mediante punteros a funciones.
- **app1.h**: archivo de cabecera central. Declara el struct "order", que es fundamental para el funcionamiento del programa, ya que administra y cataloga cada dato de las ventas contenidas en el archivo CSV. También, declara el resto de funciones utilizadas y permite la comunicación entre los distintos módulos.

Estructuras de datos

Para el almacenamiento de datos del archivo CSV, se diseñó un struct llamado “order”, con la finalidad de representar cada línea del archivo. Este struct incluye los siguientes campos relevantes:

```
struct order {
    int pizza_id;
    int order_id;
    char pizza_name[100];
    int quantity;
    char order_date[11];
    char order_time[9];
    double unit_price;
    double total_price;
    char pizza_size;
    char pizza_category[20];
    char pizza_ingredients[200];

};
```

Si bien este es el struct fundamental para el funcionamiento del programa, posteriormente se añadieron una amplia variedad de structs, para el cumplimiento de otras funciones menores. Algunas de las estructuras mencionadas son las siguientes:

- pizza_ventas: utilizada en las funciones *pms* y *pls*, con el propósito de acumular la cantidad de pizzas vendidas, según el nombre de la pizza. Se creó con la intención de almacenar los nombres de las pizzas junto con su cantidad total, para así recorrer una sola vez el arreglo de órdenes y realizar el cálculo necesario.

```
char* pms(int *size, struct order *orders) {
    struct pizza_ventas {
        char nombre[MAX_STRING];
        int cantidad;
    };
```

  Donde “MAX_STRING” fue definido como una constante (100) para facilitar el cambio de tamaño de strings en todo el programa, así manteniendo la claridad del código.

- venta_fecha y fechas: utilizada en las funciones dms, dls, dmsp y dlsp con la idea de acumular ventas totales, sea por dinero o por cantidad, por fecha. El struct permite agrupar los datos por día y encontrar los extremos (mayor o menor) con una sola pasada por los datos.

```
struct venta_fecha {
    char fecha[11];
    double total;
};
```

```
struct fecha {
    char fecha[11];
    int cantidad;
};
```

- Ingrediente: utilizada en la función *ims*, con la intención de llevar un conteo acumulado de cada ingrediente individual. Dado que los ingredientes vienen como una cadena separada por comas en el archivo CSV, esta estructura permite descomponer esa cadena y llevar la cuenta individual de cada uno de ellos.

```
struct ingrediente {
    char nombre[MAX_STRING];
    int cantidad;
};
```

  Esto fue especialmente relevante porque los ingredientes no están representados como campos separados en el archivo original, por lo que fue absolutamente necesario crear esta estructura para extraer e interpretar correctamente los datos.

- Cat_ventas: utilizado en la función *hp*, se implementó con el propósito de acumular la cantidad de pizzas vendidas por cada categoría (classic y veggie). Permite generar una estadística detallada sobre qué tipos de pizzas son más populares según su categoría.

```
struct cat_ventas {
    char nombre[30];
    int cantidad;
};
```

Justificación de las decisiones de diseño

a. Parseo del CSV:

   Se optó por una lectura manual con fgets() y strchr() para moverse campo por campo, dado que el formato del archivo es consistente y sin errores, según el enunciado. No se utilizó ninguna librería externa ni strtok() en general, salvo para los ingredientes, puesto que eso podría haber complicado el manejo de campos entrecomillados como los ingredientes.

b. Almacenamiento de ingredientes:

   Los ingredientes se almacenan como un único char llamado “pizza_ingredients[100]” en el struct order. Para calcular métricas como el ingrediente más vendido, se usa strtok() para dividir esta cadena, acumulando las cantidades en una estructura auxiliar mencionada anteriormente.

c. Uso de punteros a funciones:

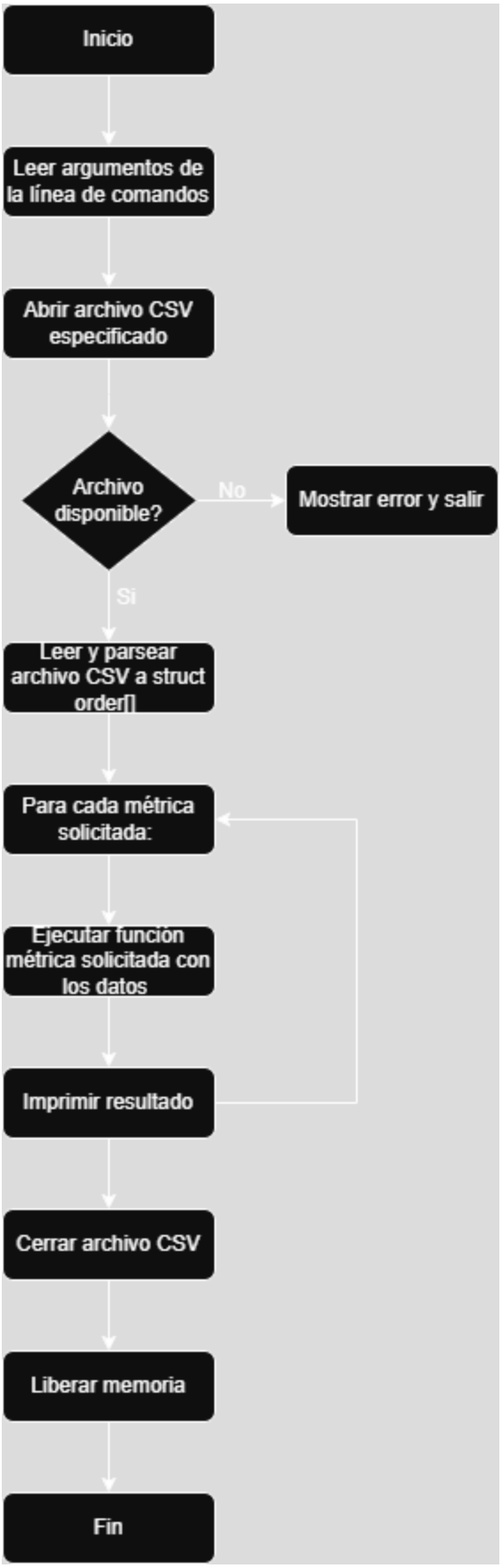
   Todas las métricas fueron implementadas con una firma común:

```
char* (*metrica)(int *size, struct order *orders);
```

   Esto permitió construir un arreglo de punteros a funciones, haciendo que el archivo main.c pueda invocar dinámicamente cualquier métrica en función del nombre ingresado por el usuario.

Interacción entre archivos

La funcionalidad general se describe de una forma fácilmente legible con el siguiente diagrama de flujo:


Inicio
Leer argumentos de la línea de comandos
Abrir archivo CSV especificado
Archivo disponible?
No
Mostrar error y salir
Si
Leer y parsear archivo CSV a struct order[]
Para cada métrica solicitada:
Ejecutar función métrica solicitada con los datos
Imprimir resultado
Cerrar archivo CSV
Liberar memoria
Fin

Detalladamente, el proceso empieza con main.c llamando a la función read_csv() definida en utils.c, para así leer el archivo CSV y cargar los datos en memoria. Luego, recorre el arreglo metrics[] (en main.c), el cual contiene los punteros a funciones definidos en metrics.c. Cada función métrica trabaja directamente sobre el arreglo de orders, y devuelve un char* con el resultado formateado. Al finalizar, main.c libera los datos mediante free_orders(), definida en utils.c, para así evitar leaks de memoria.

Uso de IA

Durante el desarrollo del proyecto se utilizó ChatGPT como herramienta de apoyo técnico, con el objetivo de discutir decisiones de diseño, optimizar fragmentos de código y principalmente, revisar errores difíciles de detectar durante la implementación. En primera instancia, la IA fue útil para decidir la estructura de software, ya que anteriormente como grupo no habíamos trabajado dividiendo el programa en distintos archivos, lo que resultó siendo tremendamente práctico para el trabajo en equipo. Luego, se utilizó también para algunas partes del código, donde pedimos sugerencias para cómo avanzar en ciertos escenarios, debido a nuestra reciente introducción al lenguaje C, habían variadas funciones de este que no conocíamos. Sin embargo, en la gran mayoría de los casos, el código entregado por ChatGPT derivó en múltiples errores, por lo que sirvió para tener una base y posteriormente progresar solucionando dichos errores. Como grupo creemos que el aspecto donde la IA ayudó crucialmente fue: solucionando errores. Específicamente en la función read_csv(), nos encontramos con un sin fin de problemas de formateo, debido al formato del archivo CSV. Tuvimos que lograr separar los datos, los cuales estaban algunos separados por comas, otros contenían comillas, etc. Sin duda concordamos que la utilización de IA agilizó de gran manera el avance del archivo utils.c.

La información obtenida fue estrictamente contrastada con la documentación oficial del lenguaje C, como también ayuda de terceros (un familiar de uno de los integrantes es informático profesional) y el apoyo de Stack Overflow para algunas dudas puntuales.

Referencias utilizadas

- Documentación oficial de C
- Stack Overflow