App 1: Pizzería en C

Ingeniería civil Informática, TICS200
Universidad Adolfo Ibáñez
Profesor: Justo Vargas
2025-S1

Integrantes:
Josefa Juica
Cristobal Zeballos
Fernando Flores

Organización del código

Escogimos usar structs para almacenar el contenido del csv, esto debido a que usa menos líneas, memoria y es más organizado.También se usó un método de parseo que consiste en detectar 2 comas consecutivas, y almacenar la palabra que esté en medio de las comas en un atributo del struct, en el caso de los ingredientes, se detecta la primera doble comillas y la última y se almacena lo que esté en medio, así hasta llegar al final de la línea, y hasta que se hayan procesado todas las líneas del código. Al almacenar los datos en structs también facilita su manejo, extracción y manipulación

El código que se utiliza tiene un enfoque que divide la lógica del programa en funciones separadas, cada una enfocada en calcular una métrica específica relacionada con las compras de pizzas. Esta forma de organización no solo mejora la legibilidad, sino también la mantenibilidad del código, lo cual es fundamental al momento de hacer un código extenso.

Uso de funciones

Cada función implementada en el código representa una tarea clara y concisa.

1. pms() (Pizza más vendida):
    - Recorre el arreglo de instancias de Compra y agrupa las cantidades vendidas por PizzaNameId. Al final, identifica la pizza con la mayor cantidad vendida y la imprime.
2. pls() (Pizza menos vendida):
    - Funciona de forma similar a pms(), pero busca el mínimo en lugar del máximo. Esto permite conocer no solo las estrellas en ventas, sino también aquellas que podrían necesitar atención o promoción.
3. dms() (Fecha con mayor dinero recaudado):
    - Agrupa las ventas totales por fecha y determina cuál tuvo el ingreso más alto. Esto es crucial para evaluar el rendimiento en días específicos.
4. dls() (Fecha con menor dinero recaudado):
    - Similar a dms(), pero se enfoca en identificar la fecha que tuvo los peores resultados financieros. Esta información es valiosa para entender períodos de baja actividad en ventas.
5. dmsp() (Fecha con mayor cantidad de pizzas vendidas):
    - Agrupa las cantidades de pizza vendidas por fecha, y luego determina cuál tuvo el mayor número. Proporciona información sobre patrones de comportamiento en días más activos.
6. dlsp() (Fecha con menor cantidad de pizzas vendidas):
    - Tiene el mismo enfoque que dmsp(), pero busca el día con ventas más bajas. Esto ayuda a detectar posibles problemas o tendencias a mejorar.
7. apo() (Promedio de pizzas por orden):
    - Suma total de pizzas vendidas y las divide entre la cantidad total de órdenes, lo cual proporciona una visión útil sobre los hábitos de compra de los clientes.

8. apd() (Promedio de pizzas vendidas por día):
    - Similar a apo(), pero este enfoque se centra en el promedio diario. Permite observar tendencias de venta diarias y comparar días entre sí.
9. ims() (Ingrediente más vendido):
    - Analiza los ingredientes de cada compra y cuenta cuántas veces aparece cada uno. Esta métrica podría ser útil para la gestión de inventario y compras.
10. hp() (Cantidad de pizzas vendidas por categoría):
    - Agrupa las ventas por categoría de pizza y muestra los totales. Esta función es clave para entender las preferencias de los clientes según categorías específicas.

El código ha sido organizado con un enfoque estructurado y modular, orientado a representar, procesar y analizar datos provenientes de un archivo CSV relacionado con compras de pizzas. Para lograr una correcta lectura de la información, se utilizan estructuras de datos definidas mediante *struct*, las cuales permiten encapsular múltiples atributos asociados a una misma entidad lógica. La estructura central del programa es *orden*, pero también se emplean otras estructuras locales dentro de funciones específicas, como *OrdenPizzas*, *VentasFecha*, *PizzaVentas* o *Ingrediente*, que cumplen funciones auxiliares específicas y temporales dentro del procesamiento.

La estructura orden es fundamental en este proyecto, ya que representa cada registro del CSV como una única unidad de información. Esta estructura contiene los siguientes campos: PizzaId, OrdenId , PizzaNameId, Cantidad, Fecha y Hora de la compra, PrecioUnitario y PrecioTotal, Tamanio, Categoría, Ingredientes y el Nombre de pizza. Al agrupar todos estos datos bajo un mismo identificador, se facilita el acceso y la manipulación ordenada de cada compra. En vez de mantener múltiples arreglos paralelos para cada columna del CSV, el uso de estructuras permite un acceso directo mediante notación de punto (compras[i].PrecioTotal, compras[i].Fecha, etc.).

El uso de punteros, aunque no se utilizan con funciones directamente, se hace un uso intensivo de punteros en el paso de arreglos de estructuras a funciones. Al enviar el arreglo *ordenes* como parámetro, lo que se pasa en realidad es un puntero al primer elemento del arreglo, lo que permite que las funciones trabajen directamente sobre los datos originales sin duplicarlos en memoria. Esto hace al programa más eficiente tanto en términos de espacio como de tiempo de ejecución. En C, este comportamiento es fundamental cuando se trabaja con grandes volúmenes de datos, como en este caso, donde cada registro contiene múltiples campos.

Diagrama de flujo

Hicimos el diagrama de flujo del código que podemos ver en la imagen 1 con la página web PlantText. Podemos ver a grandes rasgos como funciona lo explicado en el ítem anterior

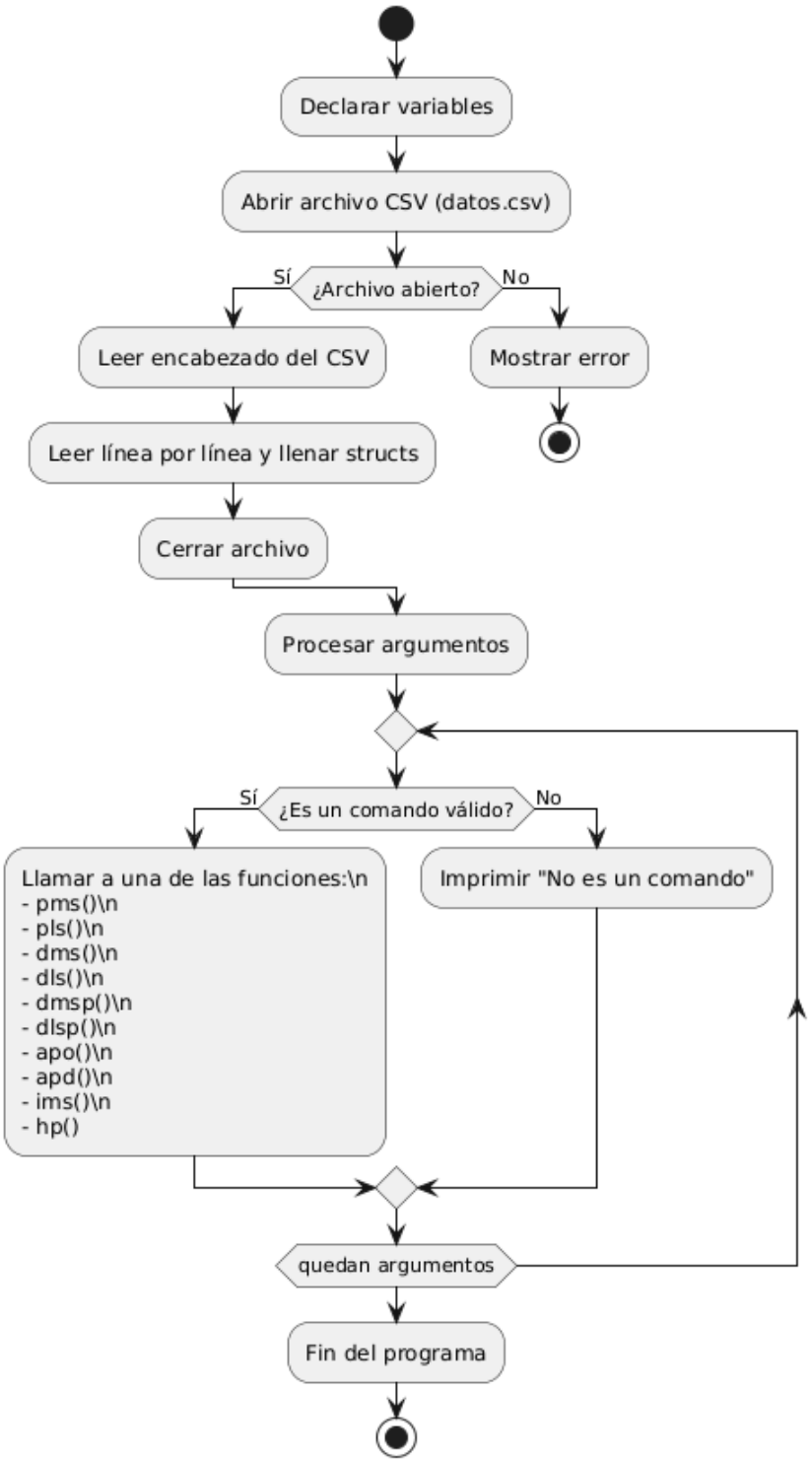
Declarar variables
Abrir archivo CSV (datos.csv)
¿Archivo abierto?
Sí
No
Leer encabezado del CSV
Mostrar error
Leer línea por línea y llenar structs
Cerrar archivo
Procesar argumentos
¿Es un comando válido?
Sí
No
Llamar a una de las funciones:\n
- pms()\n
- pls()\n
- dms()\n
- dls()\n
- dmsp()\n
- dlsp()\n
- apo()\n
- apd()\n
- ims()\n
- hp()
Imprimir "No es un comando"
quedan argumentos
Fin del programa

Imagen 1

Reflexiones finales o autoevaluación

Reflexión: Cristobal Zeballos

Lo más complejo personalmente en esta tarea fue comprender el funcionamiento de los punteros en C. Al principio resultaba confuso entender cómo una variable podía contener la dirección de otra, y cómo esto permitía modificar valores directamente desde otras funciones sin perder la información original. Sin embargo, a medida que avanzaba, me di cuenta que los punteros son muy útiles para manipular estructuras complejas, como arreglos de datos o registros leídos desde archivos, porque no dañan la función original al momento de usarlos. También al principio no pude entender el uso de funciones tipo *void*, especialmente en relación con el hecho de que no devuelven ningún valor, pero permiten ejecutar procesos importantes dentro del flujo del programa. Para enfrentar los errores y realizar pruebas, use constantemente herramientas como la inteligencia artificial y videos en YouTube que explican paso a paso conceptos que no siempre están bien detallados en la teoría. Además, algo que fue clave en nuestro aprendizaje fue utilizar un espacio de prueba adicional fuera del archivo principal en VSCode, donde podía experimentar y probar fragmentos de código antes de incorporarlos a la versión final. Esto permitió que entendiera mejor cada línea y evitar errores. Como lección principal, me quede que al trabajar con archivos CSV en C requiere una planificación detallada del uso de estructuras, un buen manejo de punteros y mucha práctica para dominar la manipulación de texto y base de datos.

Reflexión: Josefa Juica

En mi caso, tuve más problemas con el subsistema de Windows para Linux, y haciendo un pull request en github que con el código de la tarea en sí. Me costó que me funcionara el visual con el subsistema de Linux y descargar git, para mediante ubuntu hacer un pull request en github, fue muy complicado y tomó bastante tiempo. Finalmente, no pude subir mi contribución al main, me aparecía un error de que mi branch (josefa-juica) y el main tenían diferentes historias de commit, como podemos ver en el anexo (imagen 2 y 3). Por lo que mi compañero Fernando subió el código de mi branch al main. Sin embargo, se puede ver en mi branch que yo me encargué de hacer las 6 primeras métricas. Esto lo hice con ayuda de chatgpt, entendiendo qué tenía que hacer la función, qué tenía que buscar en el csv, las columnas que tenía que mirar, con eso hice promt y fui ajustando el código.

Con respecto a los errores, tuve muchos errores al tratar de leer el csv. En primer lugar traté de separar con comas el csv en el código y que la columna de los ingredientes se tomara como una sola tomando en cuenta las comillas (que empezara una columna con una comilla y termina con la otra). Me funcionó para hacer la métrica de pms, pero luego me falló, y no lo pude solucionar. Luego, mi compañero Fernando pudo lograr que los datos de las filas se almacenen en un arreglo de structs. Con esto pude hacer las 6 primeras métricas sin problema.

La lección que aprendí es que C es extremadamente preciso y puede tener errores fácilmente, es mucho más difícil de leer que lenguajes como Python y hay que definir muy bien cómo lee el csv para que no tenga errores. Cuando lee bien el csv, las métricas no son difíciles de implementar.

Reflexión: Fernando Flores

En mi opinión lo más complicado de esta tarea fue tener que aprender no solo otro lenguaje, el cual es más complicado y específico que Python, sino también tuve que cambiar mi forma lógica de pensar a la hora de codear, parecido a tener que aprender otro idioma incluido su gramática. Pese a las adversidades considero este trabajo importante y entretenido, además de trabajar habilidades blandas como el trabajo en equipo y la comunicación. Aunque sabía cómo usar github, tuve que investigar cómo trabajar en equipo cuando uno es el dueño del repositorio. Supimos enfrentar los problemas, las trabas y los bugs entre todos, cada quien sacaba su "forte" y logramos completar la tarea. En mi experiencia aprendí que codear en equipo es muy importante y necesario, pues muchas cabezas piensan mejor que una, además sería muy sacrificado hacer proyectos más grandes que este en solitario. También aprendí que C es un lenguaje muy completo, donde hay más posibilidades, especificaciones y trucos, que una vez entendidos pueden jugar a tu favor, como punteros o structs. C para mi significa un trabajo engorroso pero disfrutable, significa estar muy atento y ser muy meticuloso con lo que escribes o haces, significa cambiar tu modo de pensar y de analizar, en especial cuando trabajas con objetos tan significativos y grandes como una base de datos. Significa ser cuadrado como una computadora y pensar como una también, así como también saber cómo pedir ayuda a las mismas computadoras como Chat GPT, no con el objetivo de copiar, sino usarla de manera responsable para el aprendizaje y aplicación de contenidos, cosa que requiere un gran autocontrol y madurez.

IA

Se utilizó ChatGPT como herramienta de apoyo durante el desarrollo del programa, lo cual fue especialmente útil para entender cómo leer y parsear archivos CSV en C. También brindó ayuda en la creación y organización de funciones que trabajan con arreglos de estructuras, explicando cómo pasar datos por referencia sin perder la información original.

Otra área clave en la que la IA fue de gran ayuda fue en aprender a compilar desde la terminal usando gcc, y posteriormente entender cómo funcionan los makefiles para automatizar la compilación del proyecto. Además, fue muy útil para conectar correctamente el entorno de desarrollo VSCode con GitHub, explicando paso a paso cómo clonar repositorios, inicializar proyectos con Git, realizar commits y subir cambios al repositorio remoto. La validación de toda la información proporcionada por la IA se hizo mediante pruebas directas en VSCode, ejecutando versiones simplificadas del código para verificar que

los fragmentos propuestos funcionaban correctamente antes de integrarlos al proyecto completo. Finalmente reforzó los temas más técnicos como lo son el uso de *sscanf*, *fgets,* manipulación de cadenas y estructuras.

Referencias bibliográficas

https://www.w3schools.com/c/
https://devdocs.io/c/
https://www.reddit.com/r/C_Programming/
https://chatgpt.com/
https://www.freecodecamp.org/news/format-specifiers-in-c/

Anexo

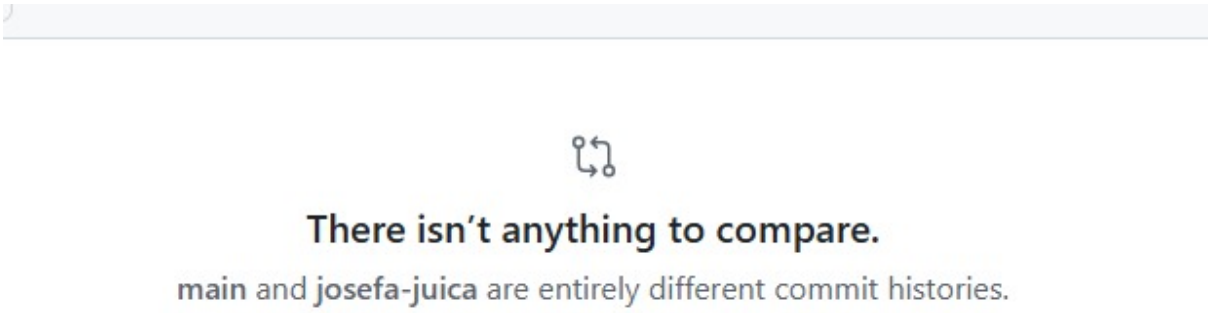

Imagen 2

```
66  66    }
67        - void pls() {
68        -     printf("Llamaste a pls\n");
      67  + void pls(Compra compras[], int registro) {
      68  +     typedef struct {
      69  +         char nombre[50];
      70  +         double total_vendida;
      71  +     } PizzaVentas;
      72  +
      73  +     PizzaVentas ventas[100];
      74  +     int total_pizzas = 0;
      75  +
      76  +     for (int i = 0; i < registro; i++) {
      77  +         int encontrada = 0;
      78  +         for (int j = 0; j < total_pizzas; j++) {
      79  +             if (strcmp(compras[i].PizzaNameId, ventas[j].nombre) == 0) {
      80  +                 ventas[j].total_vendida += compras[i].Cantidad;
      81  +                 encontrada = 1;
      82  +                 break;
      83  +             }
      84  +         }
      85  +         if (!encontrada) {
      86  +             strcpy(ventas[total_pizzas].nombre, compras[i].PizzaNameId);
      87  +             ventas[total_pizzas].total_vendida = compras[i].Cantidad;
      88  +             total_pizzas++;
      89  +         }
      90  +     }
      91  +
      92  +     // Buscar la pizza menos vendida
      93  +     double min_cantidad = ventas[0].total_vendida;
      94  +     char pizza_menos_vendida[50];
      95  +     strcpy(pizza_menos_vendida, ventas[0].nombre);
      96  +
      97  +     for (int i = 1; i < total_pizzas; i++) {
      98  +         if (ventas[i].total_vendida < min_cantidad) {
      99  +             min_cantidad = ventas[i].total_vendida;
     100  +             strcpy(pizza_menos_vendida, ventas[i].nombre);
     101  +         }
     102  +     }
     103  +
     104  +     printf("La pizza menos vendida fue: %s con un total de %.0f unidades vendidas.\n", pizza_menos_vendida, min_cantidad);
69  105  }
70        - void dms() {
71        -     printf("Llamaste a dms\n");
     106  +
     107  + void dms(Compra compras[], int registro) {
```

Imagen 3