UNIVERSIDAD ADOLFO IBÁÑEZ

Informe Tarea 1

Pizzería en C

Integrantes	Matías Astraín Javier Baeza Diego Calvo Khushang Methwani Rodrigo Sánchez
Curso	Lenguajes y Paradigmas
Sección	1
Profesor	María Loreto Arriagada
Fecha de entrega	03-04-2025

Tabla de contenido

Introducción ***2***

Objetivos 2

Desarrollo ***3***

ventas.csv 3

utilidades.c y utilidades.h 3

metrican.c y metrican.h 5

main.c 6

Conclusión ***8***

Reflexiones finales 8

Explicación de uso de la IA 9

Introducción

En esta tarea se pidió realizar un código en lenguaje C, cuyo objetivo es usar como base de datos un *CSV*, el cual tiene información de las ventas de una pizzería, y según lo que pide el usuario, debe entregar información como pizza más vendida, fecha con más ventas en términos de dinero, promedio de pizzas por orden, etc.

El archivo *CSV*, el cual debe ser creado por el grupo, debe tener las siguientes columnas: *pizza_id, order_id, pizza_name_id, quantity, order_date, order_time, unit_price, total_price, pizza_size, pizza_category, pizza_ingredients, pizza_name*.

Luego, cuando se ejecute el código, el usuario solamente deberá ingresar las métricas que necesite. Cabe destacar que el orden en el que el usuario ingresa las métricas será el mismo orden en el que el programa las mostrará (es distinto ingresar "*pms pls*" que "*pls pms*").

i. `pms` : Pizza más vendida
ii. `pls` : Pizza menos vendida
iii. `dms` : Fecha con más ventas en términos de dinero (junto a la cantidad de dinero recaudado)
iv. `dls` : Fecha con menos ventas en términos de dinero (junto a la cantidad de dinero recaudado)
v. `dmsp` : Fecha con más ventas en términos de cantidad de pizzas (junto a la cantidad de pizzas)
vi. `dlsp` : Fecha con menos ventas en términos de cantidad de pizzas (junto a la cantidad de pizzas)
vii. `apo` : Promedio de pizzas por orden
viii. `apd` : Promedio de pizzas por día
ix. `ims` : Ingrediente más vendido
x. `hp` : Cantidad de pizzas por categoría vendidas

Ilustración 1: Métricas que podría pedir el usuario

Objetivos

Los objetivos de esta tarea son aprender a estructurar un programa en varios archivos fuente de manera ordenada, lógica y modular; utilizar herramientas clásicas de apoyo a la programación como GIT, make, debuggers, etc.; utilizar punteros y, en especial, punteros a funciones; desarrollar un sistema capaz de leer un archivo CSV y aplicar sobre él varias métricas (fórmulas matemáticas simples).

Desarrollo

ventas.csv

Con ayuda de ChatGPT, se desarrolló un archivo CSV con datos de 100 ventas (creadas de forma aleatoria), incluyendo todas las columnas que mencionamos anteriormente. Con esto listo, se comenzó con el desarrollo del código.

utilidades.c y utilidades.h

Estos archivos trabajan juntos para leer el archivo CSV que contiene los datos de la pizzería y luego dejándolos almacenados dentro de un arreglo de estructura llamado *Pedido*.

Dentro de *"utilidades.h"* se define la estructura *Pedido* y otras funciones. Dentro de la estructura se almacenan las variables necesarias para pedir una pizza, como por ejemplo, ID y cantidad, que van designados con un "*int"* por ser variables enteras. Otras que son de tipo *"character"* como tamaño, ingredientes, nombre, fecha, y varios más. Finalmente tenemos variables de tipo "*float",* como el precio por unidad y precio total del pedido.

```
typedef struct {
    int pizza_id;
    int order_id;
    char pizza_name_id[75];
    // ... otros campos ...
} Pedido;
```

Luego, se declara la función *read_csv,* la cual lee el archivo *CSV* y llena los datos en la estructura.

```
int read_csv(const char *filename, Pedido *orders);
s'
void free_orders(Pedido *orders);
```

Finalmente, *#ifndef* y *#endif* aseguran que no haya repeticiones de un mismo tipo de archivo.

Siguiendo con el archivo *"utilidades.c",* aquí se pueden incluir e implementar todo lo que fue declarado en el archivo *"utilidades.h"*.

Primero se usó la función *parse_csv_line,* función que parsea una línea del CSV, respetando comillas dobles.

```c
int parse_csv_line(char *line, char **fields, int max_fields) {
    int field_count = 0;
    char *ptr = line;
    int in_quotes = 0;

    fields[field_count++] = ptr; // Primer campo

    while (*ptr) {
        if (*ptr == '"') in_quotes = !in_quotes; // Maneja comillas
        else if (*ptr == ',' && !in_quotes) {    // Separador de campo
            *ptr = '\0'; // Termina el campo actual
            fields[field_count++] = ptr + 1; // Siguiente campo
        }
        ptr++;
    }
    return field_count; // Número de campos encontrados
}
```

Después con la función *print_orders,* se pueden observar los pedidos almacenados impresos en la pantalla o terminal, una vez que se corra el código.

```c
void print_orders(Pedido *orders, int count) {
    for (int i = 0; i < count; i++) {
        printf("Pizza ID: %d\n", orders[i].pizza_id);
        // ... imprime otros campos ...
    }
}
```

Finalmente, la función read_csv, que, como se mencionó anteriormente, almacena los datos del *CSV* en la estructura *Pedido* que se definió en *"utilidades.h"*.

```c
int read_csv(const char *filename, Pedido *orders) {
    FILE *fp = fopen(filename, "r");
    // ... (manejo de errores)

    fgets(line, MAX_LINE, fp); // Ignora la primera línea (encabezados)

    while (fgets(line, MAX_LINE, fp)) { // Lee línea por línea
        char *fields[MAX_FIELDS];
        int field_count = parse_csv_line(line, fields, MAX_FIELDS);

        // Asigna cada campo a la estructura Pedido
        orders[count].pizza_id = atoi(fields[0]);
        strcpy(orders[count].pizza_name, fields[11]);
        // ... otros campos ...
        count++;
    }
    fclose(fp);
    return count; // Número de pedidos leídos
}
```

metrican.c y metrican.h

Estos dos archivos tienen el objetivo de definir y calcular las métricas que podría pedir el usuario, las cuales podrías ser cualquiera de las opciones que se encuentran en la siguiente imagen.

```c
// Métricas con sus respectivas siglas del enunciado
// 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
```

Entonces, primero se empezó con la realización del archivo *metrican.h*, donde se declararon todas las funciones que se usarán para calcular las métricas. Lo cual quedó de la siguiente manera.

```c
char* pizza_mas_vendida(int*size, Pedido *orders);
char* pizza_menos_vendida(int*size, Pedido *orders);
char* fecha_mas_ventas_dinero(int*size, Pedido *orders);
char* fecha_menos_ventas_dinero(int*size, Pedido *orders);
char* fecha_mas_ventas_pizzas(int*size, Pedido *orders);
char* fecha_menos_ventas_pizzas(int*size, Pedido *orders);
char* promedio_pizzas_orden(int*size, Pedido *orders);
char* promedio_pizzas_dia(int*size, Pedido *orders);
char* ingrediente_mas_vendido(int*size, Pedido *orders);
char* cantidad_pizzas_categoria(int*size, Pedido *orders);
```

Aquí se puede ver que todas las funciones reciben dos parámetros. Estos parámetros son *int*size*, un puntero a un número entero el cual corresponde a la cantidad de filas que tiene el *CSV*; y *Pedido *orders*, el cual es un puntero a la estructura *Pedidos* que habíamos definido en el archivo *utilidades.h*, y es por eso que al principio de este archivo se escribió *#include "utilidades.h"*, con el fin de poder usar las funciones del archivo *utilidades.h* en el archivo *metrican.h*.

Al tener listo el archivo *metrican.h*, se comenzó con el desarrollo de *metrican.c*. El objetivo de este archivo es realizar todos los cálculos necesarios para las métricas que se definieron en el archivo anterior.

Primero se realizó la función *contar_pizzas_repetidas*, cuyo fin es simplemente contar la cantidad de veces que se repite el nombre de una pizza dentro del archivo *ventas.csv*. Esta función será útil para el cálculo de las demás métricas.

Después se hicieron las funciones para hacer el cálculo de las 10 métricas, usando las funciones que se habían declarado en el archivo *metrican.h*.

main.c

main.c es el archivo que es compilado y ejecutado en el sistema. Primero es importante incluir las librerías necesarias junto con los archivos *utilidades.h* y *metrican.h* para ser utilizados por el programa en *main.c.* De esta manera, se juntan las secciones anteriores y pueden ser utilizadas en el *main.c* para obtener las métricas deseadas.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include <ctype.h>
#include "utilidades.h"
#include "metrican.h"
```

Lo siguiente es definir el puntero a las funciones que obtienen las métricas, a éste lo llamamos *funcion_metrica:*

```c
typedef char* (*funcion_metrica)(int*, Pedido*);
```

Se define un nuevo tipo de dato llamado *funcion_metrica,* el cual es un puntero a una función con dos parámetros: uno del tipo *int* y otro del tipo *Pedido* definido en *utilidades.h.* Dichas funciones son las encargadas de calcular y entregar las métricas pedidas.

Para que el programa entienda a qué función está asociado cada código (ej, *pms* a *pizza_mas_vendida)* se define una nueva estructura llamada *metrica* y se asocia cada métrica con sus respectivas funciones, de esta manera se puede compilar el programa llamando a las funciones a través de su código.

```c
Metrica metricas_disponibles[] = {
    {"pms",  pizza_mas_vendida},
    {"pls",  pizza_menos_vendida},
    {"dms",  fecha_mas_ventas_dinero},
    {"dls",  fecha_menos_ventas_dinero},
    {"dmsp", fecha_mas_ventas_pizzas},
    {"dlsp", fecha_menos_ventas_pizzas},
    {"apo",  promedio_pizzas_orden},
    {"apd",  promedio_pizzas_dia},
    {"ims",  ingrediente_mas_vendido},
    {"hp",   cantidad_pizzas_categoria}
};
```

¿Por qué llamar a las funciones por referencia en vez de paso por valor? Al hacer punteros a funciones se puede acceder directamente a éstas según el input del usuario y sin necesidad de hacer grandes bloques de if/switch; y en caso de necesitar nuevas métricas, bastaría con definirlas en *utilidades.c* sin necesidad de modificar el archivo *main.c.* Por lo tanto, el código es más fácil de trabajar.

Una vez están hechas todas las definiciones, funciones, variables y constantes, es hora de trabajar en la función *main,* la cual será la encargada de ejecutar todo lo antes mencionado a través de un compilador y un *input* dado por el usuario, por ejemplo:

```
./app1 ventas.csv pm
```

Para llevar esto a cabo, se definen en la función *main* los parámetros *argc* (cantidad de argumentos ingresados) y *argv* (strings con los argumentos provenientes del input):

```
int main(int argc, char *argv[])
```

Estos argumentos serán, en este orden: nombre del programa, datos a procesar y métricas a solicitar. En el caso de recibir menos de 3 argumentos se mostrará un mensaje de error y se entregará por defecto la métrica *pms (*pizza más vendida).

Conclusión

Reflexiones finales

Durante el desarrollo de la tarea, existieron algunos procedimientos que fueron dificultando el proceso, mayormente debido a la poca experiencia que tiene el grupo sobre la programación en el lenguaje C. Una de estas dificultades fue la creación del código para poder leer el archivo *ventas.csv,* donde ningún integrante del grupo tenía el conocimiento suficiente en programación en C como para desarrollar este código. Para solucionar esto y varios problemas más, utilizamos la ayuda de ChatGPT para que nos guíe en el proceso.

Pero, como grupo, la parte más difícil de la tarea no era en el desarrollo mismo del código, sino que ejecutarlo. En un principio, al ya tener todos los archivos de código listos, el programa funcionaba sin problema al ejecutarlo en Replit, pero, al intentar ejecutarlo en cualquier otro programa, no funcionaba. Debido a esto el grupo le pidió ayuda al ayudante, el cual después explicó cómo hacer funcionar correctamente el código, vinculando correctamente el repositorio de GitHub en Visual Studio Code.

Por último, el grupo aprendió varias lecciones al momento se realizar el código para calcular las diferentes métricas, donde se logró el objetivo de vincular la lectura del archivo *CSV* con los cálculos matemáticos simples que se exigían. El uso de punteros también estuvo presente en el desarrollo de las métricas, permitiendo acceder y modificar los valores que se encuentran en variables de otros archivos, y finalmente logrando un conjunto de archivos ordenados con el fin de responder correctamente las exigencias del usuario, cumpliendo con el paradigma procedural visto en clases.

Explicación de uso de la IA

El uso de inteligencia artificial como herramienta permitió generar datos sobre las ventas de la pizzería y comprender ciertos aspectos de mayor complejidad del lenguaje programación C. En particular se utilizó para entender mejor funciones y formas de renderizar más el código debido al conocimiento no tan avanzado del lenguaje por parte de los integrantes, ayudando a generar un orden en la realización del código, además de explicar de manera más detallada los problemas y “warnings” en la compilación de este a lo largo de la realización del proyecto adjunto.