LogiData Sudamérica

Más que envíos, decisiones inteligentes

PROYECTO PROGRAMACIÓN III

Integrantes: Guillermina Sánchez, Valentina Rey y Carola Jalil

¿Quienes somos?

LogiData Sudamérica

- Empresa joven dedicada a soluciones inteligentes para logística y ventas.
- Especialistas en análisis de datos aplicados a empresas reales.
- Misión: convertir los datos de una empresa en decisiones estratégicas.
- Visión: ofrecer tecnología clara, funcional y humana.

¿Por qué elegir a nuestra empresa?

Ventajas de LogiData como equipo de desarrollo

- Traducimos lo complejo a simple: programación entendible.
- Pensamos en la empresa real, no solo en el código.
- Medimos la eficiencia de cada proceso.
- Ofrecemos soluciones concretas: comparar, filtrar, mejorar.

¿Por qué elegir a nuestro sistema?

Ventajas del sistema de análisis de ventas

- Lee archivos CSV automáticamente.
- Ayuda a detectar patrones de ventas.

- Compara países, productos y medios de envío.
- Recalcula todo si se modifica una venta.

¿Qué hacemos?

Creamos un sistema de análisis de ventas simple, poderoso y humano

- Carga datos desde archivo CSV (una planilla).
- Procesa automáticamente la información.
- Permite consultar, comparar y modificar ventas.
- Ayuda a tomar decisiones logísticas y comerciales.

LIBRERÍAS

1. #include <iostream>

Permite trabajar con entrada y salida de datos en consola.
Se usa cout para mostrar mensajes y cin para pedir información al usuario.

2. #include "Venta.h"

Incluye la definición de la clase Venta, que es la estructura base del proyecto.
Ahí están todos los atributos de una venta y sus métodos get y set.

3. #include "OrdenadorGenerico.h"

Contiene una función genérica de QuickSort para ordenar listas con cualquier tipo de dato.
Se usa para ordenar productos, ciudades, categorías, etc., según distintos criterios.

4. #include <string>

Permite trabajar con cadenas de texto usando la clase string.
Es esencial para campos como país, producto, ciudad, etc.

LIBRERÍAS

5. #include <sstream>

Sirve para leer texto como si fuera un flujo de datos.
Se usa para procesar líneas del archivo CSV y separar los campos (con comas).

6. #include <fstream>

Permite abrir, leer y escribir archivos externos (como el archivo de ventas).
Con ifstream se lee y con ofstream se puede guardar datos.

7. #include <chrono>

Se usa para medir cuánto tiempo tarda en ejecutarse un método.
Con high_resolution_clock se puede calcular la duración de un proceso.

8. #include "ListaProyecto.h"

Define una lista simple (enlazada) personalizada.
Se usa para guardar listas de productos, países, ciudades, etc.

LIBRERÍAS

9. #include "ListaDobleProyecto.h"

Define una lista doblemente enlazada.

Se usa principalmente para manejar las ventas, ya que permite recorrer hacia adelante o atrás.

10. #include "HashMapListProyecto.h"

Define un HashMap personalizado que usa listas para manejar colisiones.

Es útil para acceder rápidamente por clave (como país → productos o fechas → montos).

11. #include "ArbolBinarioProyecto.h"

Define un árbol binario de búsqueda.

Se usa para mantener elementos ordenados alfabéticamente, como países o categorías.

12. #include <filesystem>

Sirve para verificar si un archivo o carpeta existe en el sistema antes de intentar abrirlo.

Evita errores si el archivo de ventas no está.

LIBRERÍAS

13. #define NOMBRE_ARCHIVO
"C:/Users/guill/OneDrive/Documentos/PROYECTO/ventas_sudamerica.csv"
Define una constante que guarda la ruta del archivo de ventas.
Se usa para que, si cambia la ubicación del archivo, solo haya que modificar una línea.

14. #include <stack>
Permite utilizar pilas para guardar las ventas eliminadas y podes deshacerlas en caso de equivocacion.

15. using namespace std::chrono;
Permite usar elementos como high_resolution_clock directamente, sin anteponer std::chrono::

MÉTODO 1: Top 3 ciudades con mayor monto por país

1. • **HashMapList<string, HashMapList<string, float>*>:**

Para acceder rápido al monto total de ventas por ciudad dentro de cada país.

2. • **Lista<string> paises:**

Para recorrer los países en el orden en que aparecen.

3. • **HashMapList<string, Lista<string>>:**

Para saber qué ciudades están registradas en cada país.

4. • **Lista<ciudad_monto>:**

Para agrupar ciudad + monto total y poder ordenarlas.

5. • **ordenarListaGenerica:**

Para ordenar las ciudades de mayor a menor según el monto.

6. • **mostrar_top3_recursivo:**

Para mostrar las 3 primeras ciudades ordenadas sin usar bucle.

MÉTODO 2: Monto total vendido por producto y país

1. • **HashMapList<string, HashMapList<string, float>*>:**

Permite acceder rápido al monto total de un producto dentro de cada país.

2. • **Lista<string>:**

Para recorrer los países y los productos en el orden en que se registraron.

3. • **HashMapList<string, Lista<string>>:**

Relaciona cada país con su lista de productos, evitando búsquedas costosas.

4. • **Lista<producto_monto>:**

Agrupa producto + monto total para ordenarlos fácilmente.

5. • **ordenarListaGenerica:**

Ordena productos de mayor a menor monto total.

6.

MÉTODO 3:

Promedio de ventas de cada categoría por país

1. • **Lista<pais_categorias>:**

Lista ordenada para almacenar los países y sus categorías asociadas.

2. • **Lista<categoria_monto>:**

Dentro de cada país, agrupa la suma del monto total y la cantidad de ventas por categoría.

3. • **ListaDoble<Venta>:**

Se recorre para calcular y acumular los datos por país y categoría.

MÉTODO 4:

Medio de envío más utilizado por país

1. • **ArbolBinario<envio_dato*>* envios**

puntero a un árbol binario de punteros a envio_dato (donde se guardan los medios de envío y su cantidad para ese país)

2. **ArbolBinario<pais_envios*>**

Se usa para almacenar los países, manteniéndolos ordenados alfabéticamente

3. **Venta**

Para obtener los datos de cada venta: país y medio de envío.

4.

MÉTODO 5:

Medio de envío más utilizado por categoría

1. • **ArbolBinario<envio_categoria*>:**

Guarda las categorías de productos ordenadas alfabéticamente.

2. •**HashMapList<string, int> dentro de cada categoría:**

Guarda la cantidad de veces que se usó cada medio de envío en esa categoría.

3. • **Venta:**

Contiene los datos de cada venta individual.

MÉTODO 6:

Día con mayor cantidad de ventas

1. • **HashMapList<string, float>* ventasPorDia:**

Guarda el monto total acumulado por cada fecha.

2. • **Lista<string>* fechasUnicas:**

Guarda las fechas únicas que van apareciendo.

3. • **ListaDoble<Venta>& ventas**

Contiene todas las ventas cargadas.

MÉTODO 7:

Estado de envío más frecuente por país

1

- **HashMapList<string, HashMapList<string, int>*> estadosPorPais:**

Es un mapa de países → mapa de estados de envío → cantidad.

2

- **HashMapList<string, Lista<string>*> estadosPorPais_claves:**

Guarda por cada país, una lista con los nombres de los estados de envío que se usaron.

3

- **Lista<string>* paises:**

Guarda los nombres de los países a medida que se detectan en las ventas.

MÉTODO 8:

Producto más y menos vendido en unidades

1

- **HashMapList<string, int>* cantidadPorProducto:**

Guarda cuántas veces se vendió cada producto.

2

- **Lista<string>* productos:**

Guarda los nombres de los productos distintos a medida que aparecen.

MÉTODO 9: Ranking de ventas por productos

1. · **HasMapList<string,int>* cantidadPorProducto**

   guarda la cantidad vendida de cada producto .

2. **Lista<string>*productos**

   lista con los nombres de los productos.

3. **Lista<producto_cantidad> listaRanking**

   almacenará objetos de la estructura producto_cantidad (con nombre y cantidad) para luego ordenarlos y mostrarlos.

FUNCIONES PRINCIPALES DEL SISTEMA

Cargar Ventas

¿Para qué se usa?
Carga todas las ventas desde un archivo CSV y las convierte en objetos de tipo Venta.

Estructuras utilizadas:
- ListaDoble<Venta>: almacena todas las ventas cargadas.
- Venta: clase que contiene los datos de cada venta.

Agregar Venta

¿Para qué se usa?
Permite que el usuario cargue manualmente una nueva venta con todos sus campos.

Estructuras utilizadas:
- ListaDoble<Venta>: se agrega la nueva venta al final.

FUNCIONES PRINCIPALES DEL SISTEMA

Modificar Venta

¿Para qué se usa?
Busca una venta por su ID y permite cambiar sus datos.

Estructuras utilizadas:
- ListaDoble<Venta>: se recorre hasta encontrar la venta.
- Venta: se modifican los datos usando los métodos set.

Eliminar Ventas por Ciudad o País

Elimina todas las ventas que pertenezcan a una ciudad o país especificado.
Guarda cada venta eliminada en una pila para poder deshacer la acción.

Estructuras utilizadas:
- ListaDoble<Venta>: se recorren y eliminan ventas.
- Pila<Venta>: guarda las ventas eliminadas.

FUNCIONES PRINCIPALES DEL SISTEMA

Deshacer Eliminación

¿Para qué se usa?
Restaura la última venta eliminada desde la pila.

Estructuras utilizadas:
- Pila<Venta>: contiene las ventas que se eliminaron.
- ListaDoble<Venta>: se vuelve a insertar la venta.

Procesar Todo

Ejecuta todas las estadisticas automáticamente: top ciudades, productos, medios, fechas, etc.

Estructuras utilizadas:
- Todas las de los métodos (listas, hashmaps, árboles, etc.).
- high_resolution_clock: mide el tiempo de ejecución.

FUNCIONES PRINCIPALES DEL SISTEMA

Guardar ventas en el CSV

¿Para qué se usa?
Guarda todas las ventas registradas en un archivo CSV, actualizando su contenido con los datos actuales del sistema.

Estructuras utilizadas:
- ListaDoble<Venta>: contiene todas las ventas cargadas en memoria.
- ofstream: permite escribir en archivos de texto externo.

CONSULTAS DINÁMICAS

1. El listado de ventas realizadas en una ciudad específica.

¿Para qué se usa?
Para mostrar todas las ventas registradas en una ciudad específica ingresada por el usuario.

Estructuras utilizadas:

- ListaDoble<Venta>: almacena las ventas.
- Venta: representa cada venta con sus datos.
- for + if: recorren la lista y comparan ciudades.
- try-catch: manejan errores como ciudad vacía.
- high_resolution_clock: mide el tiempo de ejecución.
- contador_if: cuenta cuántas veces se evaluó el if.

2. El listado de ventas realizadas en un rango de fechas por país.

¿Para qué se usa?
Muestra todas las ventas realizadas en un país dentro de un rango de fechas ingresado por el usuario.

Estructuras utilizadas:

- ListaDoble<Venta>: lista de ventas.
- Validación de fechas: formato DD/MM/AAAA y existencia real.
- Conversión de fechas a AAAAMMDD para comparar fácilmente.
- for + if: filtran por país y rango de fechas.
- try-catch: manejan errores como fechas mal ingresadas.
- high_resolution_clock: mide el tiempo de ejecución.
- contador_if: cuenta cuántas veces se evaluó el if.

3. Comparación entre dos países:

a. monto total de ventas
b. productos más vendidos
c. medio de envío más usado.

¿Para qué se usa?

Comparar ventas totales, producto más vendido y medio de envío más usado entre dos países.

Estructuras utilizadas:

- ListaDoble<Venta>: contiene todas las ventas.
- HashMapList<string, float/int>: acumula montos, cantidades y usos por clave (producto o medio).
- for + if: recorren y comparan según país.
- try-catch: maneja errores sin cortar el programa.
- chrono: mide el tiempo total.

4. Comparación entre dos productos discriminado por todos los paises:

a. cantidad total vendida
b. monto total

¿Para qué se usa?

Compara la cantidad y el monto total vendidos de dos productos en todas las ventas registradas.

Estructuras utilizadas:

- ListaDoble<Venta>: contiene las ventas.
- for + if: recorren y comparan productos.
- Acumuladores: para sumar cantidad y monto por producto.
- try-catch: evita que errores detengan el programa.
- chrono: mide el tiempo de ejecución.
- contador_if: cuenta comparaciones realizadas.

5.- Buscar productos vendidos en promedio por debajo de un monto total especificado por el usuario (umbral) y por paìs.

¿Para qué se usa?
Muestra los productos vendidos en un país cuyo precio promedio por unidad es menor a un valor umbral ingresado por el usuario.

Estructuras utilizadas:

- ListaDoble<Venta>: lista de ventas.
- HashMapList<string, producto_acumulado>: acumula cantidad y monto por producto.
- Lista<string>: guarda nombres únicos de productos.
- for + if: recorren ventas y filtran por país.
- try-catch: maneja errores al acceder al hash.
- chrono: mide tiempo.
- contador_if: cuenta comparaciones.

6. Buscar productos vendidos en promedio por encima de un monto total especificado por el usuario (umbral)

¿Para qué se usa?
Muestra productos de un país cuyo precio promedio por unidad es mayor a un valor umbral.

Estructuras utilizadas:

- ListaDoble<Venta>: lista de ventas.
- HashMapList<string, producto_acumulado>: guarda cantidad y monto por producto.
- Lista<string>: lista de nombres únicos de productos.
- for + if: recorren y filtran por país.
- try-catch: maneja errores al acceder al hash.
- chrono: mide el tiempo.
- contador_if: cuenta comparaciones.

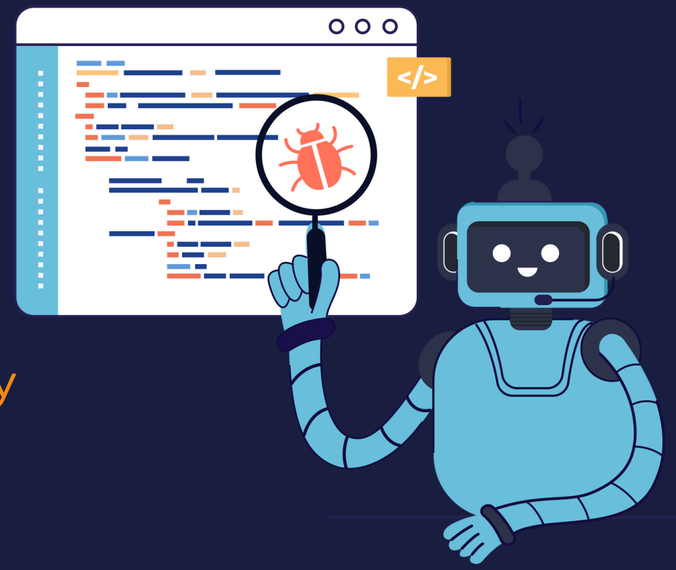
MENÚ DE OPCIONES

1. Cargar archivo de ventas
2. Agregar venta
3. Eliminar venta por ciudad o país
4. Modificar venta por ID
5. Mostrar ventas en una ciudad específica
6. Mostrar ventas en un rango de fechas por país
7. Comparar dos países
8. Comparar dos productos (cantidad y monto total)
9. Buscar productos con promedio MENOR a umbral en un país
10. Buscar productos con promedio MAYOR a umbral en un país
11. Deshacer última eliminación
0. Salir

Seguridad y control – Manejo de Excepciones

¿Qué pasa si algo sale mal?

- Usamos try/catch, una herramienta de C++ que detecta errores sin que el programa se corte.
- Sirve para controlar situaciones como:
  - Que no exista un país o ciudad buscada.
  - Que falten datos en el archivo CSV.
  - Que el usuario intente modificar una venta inexistente.
  -
- Cuando algo falla, mostramos un mensaje claro y seguimos con el programa.



¿Por qué usamos punteros?

- Crear estructuras dinámicas
- Evitar copias innecesarias
- Optimizar eficiencia y escalabilidad del sistema

Conclusión

- Sistema pensado para usuarios reales:
- Flexible, seguro e interactivo
- Automatiza análisis de ventas
- Optimizado para decisiones rápidas
- No es solo un programa, es una herramienta de inteligencia comercial

Estructura	¿Para qué se usa?	Ventajas
Lista	Recorrido secuencial, mantener orden de inserción.	Fácil de implementar y usar; mantiene orden.
ListaDoble	Recorrido bidireccional, útil para deshacer operaciones.	Permite recorrer hacia atrás; más flexible.
HashMapList	Acceso rápido por clave (país, producto, estado...).	Acceso promedio O(1); muy eficiente.
ArbolBinario	Mantener datos ordenados y buscar eficientemente.	Inserciones y búsquedas O(log n); recorrido ordenado.
Structs Auxiliares	Agrupar datos relacionados como nombre y monto.	Facilitan ordenamiento y presentación de resultados.
Funciones Genéricas	Ordenar listas de cualquier tipo de dato.	Código reutilizable y adaptable.
Recursividad	Mostrar datos como Top 3 sin usar bucles tradicionales.	Mejora comprensión de estructuras.

AHORA.. UNA PRUEBA PILOTO

g++ ProyF.cpp Venta.cpp -o output/ProyF.exe

output\ProyF.exe

g++ ProyFCorregido.cpp Venta.cpp -o output/ProyFCorregido.exe

output\ProyFCorregido.exe

¡MUCHAS GRACIAS!