**ANALISIS DEL DESAFIO II**  
  
ALEJANDRO NARANJO NARANJO   
  
1041440121   
  
**INFORME DEL DESAFÍO II: SISTEMA DE COMERCIALIZACIÓN DE COMBUSTIBLE TERMAX**

**1. ANÁLISIS DEL PROBLEMA Y SOLUCIÓN PROPUESTA**El presente desafío consiste en desarrollar un sistema para gestionar una red de estaciones de servicio de combustible para la empresa TerMax. El objetivo es automatizar las operaciones clave de las estaciones de servicio, tales como la gestión de tanques, surtidores, y las transacciones de venta de combustible.

Contexto

Una estación de servicio consta de varias componentes como tanques de combustible, surtidores, y las transacciones que ocurren en cada surtidor. El sistema debe permitir agregar, eliminar y gestionar estaciones de servicio a nivel nacional, así como simular ventas de combustible y detectar posibles fugas.

Requisitos clave

* Gestión de la red nacional: Se necesita la capacidad de agregar y eliminar estaciones de servicio, calcular el monto de ventas por categoría de combustible y fijar los precios del combustible por región.
* Gestión de surtidores: Dentro de cada estación, el sistema debe gestionar surtidores, consultar transacciones, y simular ventas de combustible.
* Verificación de fugas: Debe ser posible detectar fugas en las estaciones verificando que lo vendido y almacenado corresponde a más del 95% de la capacidad original del tanque.

El análisis del problema se enfoca en representar de manera eficiente las entidades mencionadas, aprovechando la POO para estructurar el sistema de manera modular, facilitando la gestión de la red de estaciones de servicio.

**2. DIAGRAMA DE CLASES**

|  |
| --- |
| **Surtidor** |
| - id: int - capacidad: double - combustibleAlmacenado: double  - transacciones: ListaTransacciones |
| + venderCombustible()  + obtenerID(): int + consultarTransacciones() |

|  |
| --- |
| **EstacionDeServicio** |
| -id: int  -nombre: string ubicacion: string -surtidores: Surtidor[12] tanques:  -Tanque ventasTotales: double |
| + agregarSurtidor() + eliminarSurtidor() + verificarFugas() + simularVenta()  + obtenerID(): int + obtenerCombustibleVendido(): double  +obtenerCombustibleAlmacenado(): double  +obtenerCapacidadOriginal(): double |

|  |
| --- |
| **ListaTransacciones** |
| - cabeza: NodoTransaccion\* |
| + agregarTransaccion() |

|  |
| --- |
| **NodoTransaccion** |
| - transaccion: Transaccion  - siguiente: NodoTransaccion\* |
|  |

|  |
| --- |
| **Tanque** |
| - capacidadOriginal: double  - combustibleAlmacenado: double |
| +obtenerCombustibleAlmacenado(): double  + agregarCombustible(double)  + extraerCombustible(double) |

|  |
| --- |
| **Transaccion** |
| - id: int - fecha: Date - litros: double -categoria:CategoriaCombustible |
| + obtenerDetalles() |

**3. ESTRUCTURAS DE DATOS**

Dado que no se permite utilizar ni **mapas** ni **vectores**, las estructuras de datos serán reemplazadas de la siguiente manera:

* **Arrays estáticos**: Se utilizarán arrays estáticos para gestionar las estaciones de servicio y los surtidores, con un tamaño máximo predeterminado. Por ejemplo, se puede definir un límite máximo de estaciones o surtidores que el sistema podrá manejar.
* **Listas enlazadas**: Para almacenar transacciones y gestionar la venta de combustible, se usarán listas enlazadas, que permiten añadir elementos de forma dinámica sin necesidad de vectores.

**ESTRUCTURA DE DATOS ALTERNATIVA:**

1. **Arrays estáticos**:
   * EstacionDeServicio estaciones[100]; // Un máximo de 100 estaciones en la red.
   * Surtidor surtidores[12]; // Un máximo de 12 surtidores por estación de servicio.
2. **Listas enlazadas**: Para manejar las transacciones, puedes implementar una **lista enlazada** que almacene cada venta en un nodo.

struct NodoTransaccion {

Transaccion transaccion;

NodoTransaccion\* siguiente;

};

class ListaTransacciones {

NodoTransaccion\* cabeza;

public:

ListaTransacciones() : cabeza(nullptr) {}

void agregarTransaccion(const Transaccion& nuevaTransaccion) {

NodoTransaccion\* nuevo = new NodoTransaccion;

nuevo->transaccion = nuevaTransaccion;

nuevo->siguiente = cabeza;

cabeza = nuevo;

}

};

Este enfoque evita el uso de std::vector y permite gestionar las transacciones de manera dinámica.

**4. ALGORITMOS Y SU INTRA-DOCUMENTACIÓN**

**4.1 VERIFICACIÓN DE FUGAS DE COMBUSTIBLE**

El algoritmo sigue la misma lógica, pero ahora la estructura de datos de las estaciones y surtidores será un array estático. El proceso será recorrer los arrays para encontrar la estación de servicio seleccionada y realizar la verificación.

**Código pseudocódigo con arrays:**

bool verificarFugas(EstacionDeServicio estaciones[], int numEstaciones, int idEstacion) {

for (int i = 0; i < numEstaciones; ++i) {

if (estaciones[i].obtenerID() == idEstacion) {

for (CategoriaCombustible categoria : {Regular, Premium, EcoExtra}) {

double vendido = estaciones[i].obtenerCombustibleVendido(categoria);

double almacenado = estaciones[i].obtenerCombustibleAlmacenado(categoria);

double capacidadOriginal = estaciones[i].obtenerCapacidadOriginal(categoria);

if ((vendido + almacenado) < (0.95 \* capacidadOriginal)) {

return true; // Hay una fuga

}

}

}

}

return false; // No hay fugas

}

**4.2 SIMULACIÓN DE VENTAS**

El algoritmo para simular ventas también será similar, pero al utilizar arrays para las estaciones y surtidores, se accederá a los elementos mediante sus índices en lugar de vectores o mapas. Las transacciones se almacenarán en una lista enlazada.

**Código pseudocódigo para simulación de ventas con arrays:**

void simularVenta(EstacionDeServicio estaciones[], int numEstaciones, int idEstacion) {

// Buscar la estación de servicio correspondiente

for (int i = 0; i < numEstaciones; ++i) {

if (estaciones[i].obtenerID() == idEstacion) {

Surtidor surtidor = estaciones[i].seleccionarSurtidorAleatorio();

int litros = rand() % 18 + 3; // Genera un número aleatorio entre 3 y 20 litros

CategoriaCombustible categoria = seleccionarCategoriaAleatoria();

if (surtidor.venderCombustible(categoria, litros)) {

cout << "Venta realizada con éxito." << endl;

} else {

cout << "No hay suficiente combustible. Venta parcial realizada." << endl;

}

break;

}

}

}

**5. PROBLEMAS ENFRENTADOS**

**5.1 LIMITACIÓN EN EL USO DE ARRAYS**

El principal desafío fue la **limitación en la flexibilidad** del sistema debido al uso de arrays estáticos, que requieren definir límites máximos para el número de estaciones, surtidores y transacciones. Esto obliga a anticipar el tamaño del problema en lugar de tener una estructura más dinámica como un vector.

**5.2 GESTIÓN MANUAL DE LISTAS ENLAZADAS**

Otro reto fue la implementación de listas enlazadas para gestionar las transacciones, que, aunque eficientes para agregar elementos de manera dinámica, requieren un manejo manual de la memoria, lo que aumenta la complejidad del código.

**6. EVOLUCIÓN DE LA SOLUCIÓN**

La solución inicial empleaba estructuras como vectores y mapas para facilitar la gestión de datos. Al no poder usar estas estructuras, se implementaron arrays estáticos y listas enlazadas para cumplir con los requisitos. Este cambio trajo algunos desafíos, como la gestión manual de memoria y la anticipación de los límites de almacenamiento, pero permitió desarrollar una solución eficiente dentro de las restricciones impuestas.