**DESAFIO II**



UNIVERSIDAD DE ANTIOQUIA

FACULTAD DE INGENIERÍA DE TELECOMUNICACIONES

Karol Geraldine Cardona Gil

Jhorys Osnaider Goez Rentería

DOCENTE:

Augusto Salazar Jiménez

2024

#### **Análisis del problema**

El problema consiste en gestionar una red nacional de estaciones de servicio de combustibles que incluye múltiples estaciones, cada una con surtidores que distribuyen tres tipos de combustible (Regular, Premium y EcoExtra). El sistema debe permitir la simulación de ventas, la gestión de inventario, la verificación de fugas, y ofrecer un menú para la interacción del usuario con estas funcionalidades.

Las funcionalidades a implementar se dividen en cuatro áreas principales:

1. **Gestión de la red nacional**:Para la administración de las estaciones de servicio, incluyendo agregar, eliminar y calcular ventas.
2. **Gestión de estaciones de servicio**: Se manejan los surtidores, consulta de transacciones, capacidad de tanques y ventas de combustible.
3. **Verificación de fugas**:Se hará la comprobación para detectar posibles pérdidas de combustible.
4. **Simulación de ventas**: Se hará la realización de ventas simuladas, actualizando el combustible disponible y registrando la venta.

### **Consideraciones para la solución propuesta:**

### **Eficiencia:** Se debe utilizar referencias en lugar de copias innecesarias para manejar las instancias de las estaciones de servicio, surtidores y transacciones.

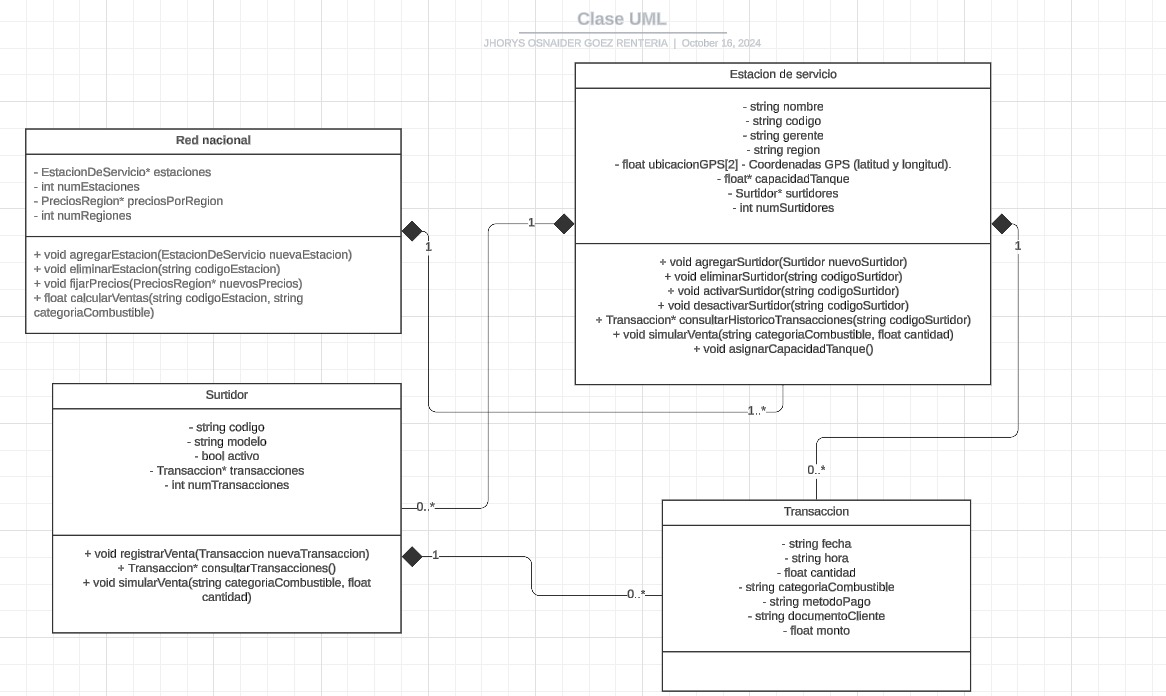
### **Uso de memoria dinámica:** Se gestionarán las listas de estaciones y surtidores utilizando memoria dinámica para permitir un crecimiento y ajuste eficientes del tamaño de las estructuras.

### **Uso de estructuras propias:** Se desarrollarán estructuras de datos personalizadas (como listas y mapas) que no dependen de la STL, para cumplir con las restricciones establecidas por el cliente.

### **subprogramas**

1. **Simulación de ventas**
   * Se selecciona aleatoriamente un surtidor activo.
   * Se genera una cantidad aleatoria de litros a vender.
   * Se verifica si hay suficiente combustible disponible, ajustando la venta si es necesario.
   * Se registra la venta y se actualiza la disponibilidad del tanque.
2. **Verificación de fugas de combustible**
   * Se suma la cantidad de combustible vendido y el almacenado en el tanque.
   * Se compara con la capacidad original del tanque para verificar que se mantenga dentro del 95%.
   * Si se detecta una discrepancia, se reporta una posible fuga.

**Diagrama de clases**



**Algoritmos Implementados Debidamente Intra-Documentados**

En esta sección se describen los algoritmos implementados en el proyecto, haciendo especial énfasis en la documentación interna del código.

**1. Algoritmo para Agregar Estaciones de Servicio**

**Descripción**

Este algoritmo se utiliza para agregar nuevas estaciones de servicio a la red nacional de estaciones. Se asegura de que siempre haya suficiente espacio para almacenar las estaciones, expandiendo el tamaño del arreglo dinámico si es necesario.

* **Validación del espacio disponible:** El método primero verifica si el arreglo dinámico tiene espacio suficiente para agregar la nueva estación.
* **Expansión del arreglo:** Si no hay espacio disponible, se llama a una función para expandir el tamaño del arreglo dinámico.
* **Adición de la nueva estación:** Una vez que hay espacio, se agrega la estación al arreglo y se actualiza el contador de estaciones.

**2. Algoritmo para Calcular Ventas por Estación de Servicio**

**Descripción**

Este algoritmo calcula el monto total de las ventas para una estación de servicio específica, basándose en la categoría de combustible seleccionada.

* **Inicialización:** El total de ventas se inicializa en cero.
* **Búsqueda de la estación:** Se recorre el arreglo de estaciones para encontrar la que corresponde con el código proporcionado.
* **Cálculo del monto:** Si se encuentra la estación, se calcula el monto total de ventas para la categoría de combustible especificada.

**3. Algoritmo para Registrar Transacciones en Surtidores**

**Descripción**

Este algoritmo utiliza la librería ctime para registrar la fecha y hora de una transacción de manera precisa.

* **Uso de ctime:** Se utiliza la función time y localtime para obtener la fecha y hora actuales del sistema.
* **Formato de la fecha y hora:** Los valores se formatean para almacenarlos en los atributos fecha y hora de la transacción, asegurando un registro preciso de la misma.

**Evolución de la solución y consideraciones para tener en cuenta en la implementación**.

**1. Evolución del Código**

* **Diseño inicial**: El diseño comenzó con la creación de las clases fundamentales: Transaccion, Surtidor, EstacionDeServicio y RedNacional. Estas clases fueron creadas para capturar la estructura básica y la relación entre los elementos del sistema.
* **Gestión dinámica de memoria**: Una de las mejoras que realizamos fue la transición al uso de punteros y arreglos dinámicos para manejar listas variables de transacciones, surtidores y estaciones. Esta adaptación permite que el sistema se expanda y maneje datos dinámicos de manera eficiente.
* **Uso de la librería ctime**: Hicimos uso de la librería estándar de C++ ctime para gestionar y registrar la fecha y hora de cada transacción automáticamente. Esta funcionalidad asegura que todas las transacciones estén temporalmente etiquetadas, lo cual es fundamental para el análisis y reporte de ventas.

**Problemas de desarrollo que se afrontaron:**

1. **Manejo de Errores y Validaciones:** Tratamos de que los datos ingresados por los usuarios fueran validados antes de guardar, asegurándonos de que los códigos de estación y los nombres tuvieran el formato correcto y no estuvieran vacíos, lo que ayudó a evitar errores en las operaciones.
2. **Simulación de Ventas:** Al generar ventas aleatorias, nos aseguramos de que los valores de combustible vendido y almacenado se registraran correctamente, lo que resultó ser un poco complicado en la codificación, ya que debíamos mantener un equilibrio entre la cantidad vendida y la disponible.
3. **Detección de Fugas:** Implementamos la lógica para detectar fugas, calculando las diferencias entre el total vendido y el almacenado, lo cual requirió atención y precisión para evitar falsas alarmas o pasar por alto problemas reales.
4. **Manejo de Memoria:** La gestión adecuada de memoria dinámica fue crucial, asegurándonos de liberar correctamente la memoria para evitar fugas de memoria, lo que podría haber comprometido el rendimiento del sistema.
5. **Integración de Funcionalidades:** Garantizar que todas las partes del sistema se integraran correctamente y funcionaran de manera coherente fue un desafío, especialmente cuando se añadieron más funcionalidades, ya que debíamos asegurarnos de que no hubiera conflictos entre ellas.