Desafío II

Entregado a: Aníbal Guerra

Augusto Salazar

Entregado por: Adriana Erazo

Universidad de Antioquia

Facultad de Ingeniería

Informática II

Medellín

2024

Tabla de contenido:

-Análisis del sistema de comercialización de combustible TerMax

-Diagrama de notación UML simplificado

**Análisis del sistema de comercialización de combustible TerMax**

El desafío II de informática II se trata de un programa en c++ de POO, en donde se solicita simular una red sobre como seria el manejo de ventas de una empresa que suministra gasolina a todo el país. El reto se trata de desde el manejo de la empresa poder crear y eliminar estaciones de gasolina, además, a las estaciones de gasolina se les desglosa una serie de tareas, como es verificar fugas de combustible, además, ver los datos de transacción cuando se realiza una venta, maneja la cantidad de surtidores, entre otras.

**Características:**

Un nombre

**Puede hacer:**

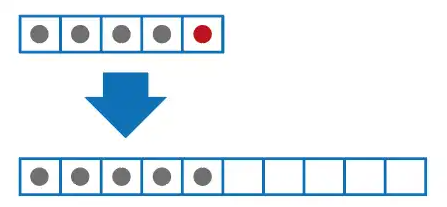
Agregar estaciones

Eliminar estación

Conocer el monto total de ventas del país

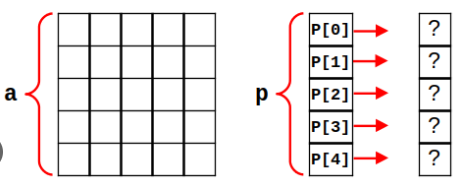
Fijar los precios de las categorías de combustible

La empresa se va a manejar con arreglos dinámicos, la lógica es poder almacenar la información que se necesita por medio de arreglos, en la creación de estaciones, no se tiene certeza de cuantas van a ser, por lo tanto, se colocara un numero especifico como máxima capacidad del arreglo para almacenar las estaciones, dado el caso de que este se llene, se multiplicara el numero anterior de capacidad por dos, para tener el doble de capacidad, entonces, se pasaran los datos ya obtenidos al arreglo nuevo y se eliminara el espacio de memoria del arreglo anterior, ya que no se lo va a usar.

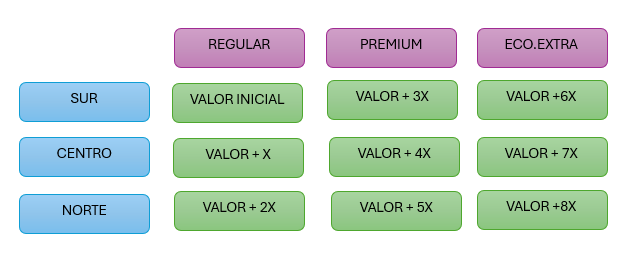


Para eliminar una estación en especifico se debe verificar antes, que no exista ningún surtidor activo, lo que conlleva a verificar que el tanque de almacenamiento del combustible este vacío, solo así se puede comprobar que todos los surtidores están inactivos. Cuando se elimine la estación, deberá eliminarse del arreglo en donde estaba almacenada.

Para conocer el monto total de ventas, en el arreglo principal de la memoria, estarán almacenados todos los datos de cada estación, la idea es usar un puntero que recorra el arreglo y busque las ventas de cada categoría de la estación, mostrar en pantalla cada estación con su respectiva venta por categorías, mostrar la suma total de las ventas por estación y, por último, un total de todas las ventas de todas las estaciones que haya en el país. Mas o menos como se indica en la imagen P[0] me dirige a la posición 0 del arreglo, y P[0][1] suponiendo que la posición de ventas de la categoría regular este en la posición 1 de la estación de la posición 0, toma ese dato y así con los que sigue de la misma estación y la imprime.



Para fijar el precio del país se realiza con un ciclo for, tomando por referencia un valor cualquiera que se le ira sumando una cantidad fija (x), para organizarlos en una “matriz 3x3”, para cada región y categoría, como se muestra en la siguiente imagen:



Para definir la capacidad el tanque de combustible se hace de manera automática, se define un valor aleatorio entre 100 y 200 litros por categoría. Se usan las librerías:

**Características:**

Código

Nombre

Región

Gerente

Ubicación GPS

**Puede hacer:**

Definir la capacidad del tanque de combustible

Define número de islas

Elimina/Crea surtidor

Desactivar/Activar surtidor

Consultar historial transacciones

Simular una venta de combustible.

Actualizar disponibilidad de combustible

Verificación de fuga de combustible

#include <cstdlib> // biblioteca de propósito general de C

que contiene la función rand () que escoge un valor entre un rango de números definido y

#include <ctime> // biblioteca de tiempo de C

que contiene la función srand (time ()) que por medio de la hora del tiempo, genera números aleatorios distintos, sin las dos funciones, sería complicado generar esto.

Para definir el numero de islas hay que tener en cuenta el numero de surtidores, ya que, la isla debe contener al menos un surtidor y la estación tiene de 2 a 12 surtidores, por consiguiente se puede decir que una estación puede tener un máximo de 12 islas, hay que tener en cuenta que para agregar una isla, hay que verificar de que no hayan ya las suficientes y dado el caso de crear una nueva, se deben redistribuir los surtidores ya existentes si ya hay 12 de ellos, si por el contrario, no los hay, no habría problema para crear una isla nueva con un nuevo surtidor.

Para eliminar y crear el surtidor, primero hay que ubicarse en la estación en específico, eso se hace por medio del código y por medio de un puntero que buscara el código de la estación, una vez ubicados ahí, la idea es que se despliegue la pregunta de eliminar o crear surtidor, si es crear, se debe verificar si se puede hacer, viendo si hay menos de 12 surtidores en la estación y para eliminar un surtidor, hay que verificar si es el único en la isla, para ver si se elimina la isla también, de lo contrario se elimina simplemente, siempre y cuando se compruebe que queden 2 surtidores como mínimo por estación.



La estación puede activar y desactivar surtidores de manera independiente, por cualquier razón podría hacerlo, simplemente hay que tener en cuenta, cuantos surtidores están activos, para mostrarle al cliente, de cual de los surtidores puede realizar su compra.

La consulta del historial de transacciones se realiza desde una estación en específico, en donde se desplegará cada categoría con fecha, hora, cantidad de litros de combustible, método de pago y precio. Estos datos se guardarán en cada estación, pero el surtidor en específico es quien tomará estos datos, después se consultará para imprimir.

Cuando se quiera simular una venta, primero se va a especificar la estación, después se comprobará cual surtidor esta activo, se procederá a pedir la cantidad de litros, el método de pago y la categoría de gasolina, después se procederá a cobrar y todos estos se procesaran, para enviar el reporte completo con fecha, hora y demás para que se guarden en la estación. Si el cliente pide una cantidad mayor a la que contiene el tanque, se vende lo que hay en el tanque y se realiza la transacción total. Después se podrá visualizar la transacción con la fecha, hora, cantidad de litros vendidos, método de pago y el monto.



Para actualizar el tanque, habría que buscar el valor que tiene el tanque y a ese, restarle la cantidad vendida, para hacerlo, se busca el valor del tanque en el arreglo de cada estación y se lo va cambiando según las ventas, así el tanque se ira actualizando hasta que se vacíe por completo.

La fuga de combustible se debe hacer con dos valores, teniendo el valor total de litros vendidos por una categoría en específico de una estación en específico, más la cantidad de litros que hay en esa categoría, después, hay que hacer una comparación de la cantidad total que tiene el tanque, si la suma de todas las cantidades es igual o mayor al 95% de la cantidad máxima que puede almacenar el tanque, se debe decir que no hay fugas.

**Diagrama de notación UML simplificado**

|  |
| --- |
| **Empresa** |
| **- std::string nombreEmpresa;**  **- float preciosCombustible[3][3];**  **- int numeroEstaciones\_actual;**  **- int numeroEstaciones\_total;**  **- Estaciones\* arregloEstaciones;**  **+ float ventasTotales;** |
| **+ void crearEstacion** **(std::string \_nombreEstacion, int \_codigoEstacion, std::string \_gerente, int \_region, float \_ubicacionGPS[2]);**  **+ void eliminarEstacion(int indice);**  **+ int getnumeroEstaciones\_actual();**  **+ string getnombreEmpresa();**  **+ int calculoMontoTotal();** |

****

|  |
| --- |
| **Estaciones** |
| **- std::string nombreEstacion;**  **- int codigoEstacion;**  **- std::string gerente;**  **- int region;**  **- float ubicacionGPS[2];**  **- float preciosCombustible[3][3];**  **- int capacidad\_tanque[3];**  **+ float historial\_Ventas[4];**  **+ float \*historial\_Transacciones;**  **+ int numero\_ventas;** |
| **+ int getregion();**  **+void agregarSurtidor(int codigoSurtidor, std::string modeloSurtidor);**  **+ void eliminarSurtidor(int codigoSurtidor, std::string modeloSurtidor);**  **+ void calcularVentasPorCategoria();**  **+ void fijarPreciosCombustible();**  **+ void actualizarDisponibilidadCombustible();**  **+ void simularVentaCombustible();**  **+bool verificarFugaCombustible();** |

|  |
| --- |
| **Surtidores** |
| **- int codigoSurtidor;**  **- std::string modeloSurtidor;**  **- bool surtidorActivo;** |
| **+ void ventaSurtidor(float \_metodo\_pago, float \_cant\_L, int tipo\_gal);** |