#include <iostream>

#include "Red.h"

#include "Linea.h"

#include "Estacion.h"

int main() {

// Creación de la red de metro.

Red redMetro("Red de Metro");

int opcion; // Variable para almacenar la opción seleccionada por el usuario.

do {

// Mostrar el menú de opciones

std::cout << "\n==== Simulador de Red de Metro ====\n";

std::cout << "1. Agregar línea\n";

std::cout << "2. Eliminar línea\n";

std::cout << "3. Agregar estación\n";

std::cout << "4. Eliminar estación\n";

std::cout << "5. Verificar existencia de estación\n";

std::cout << "6. Contar estaciones en una línea\n";

std::cout << "7. Contar estaciones en la red\n";

std::cout << "8. Calcular tiempo de llegada\n";

std::cout << "0. Salir\n";

std::cout << "Ingrese una opción: ";

std::cin >> opcion;

switch (opcion) {

// Agregar línea

case 1: {

std::string nombre;

std::cout << "Ingrese el nombre de la línea de metro: ";

std::cin >> nombre;

Linea\* nuevaLinea = new Linea(nombre, false);

redMetro.agregarLinea(nuevaLinea);

std::cout << "Línea creada.\n";

break;

}

// Eliminar línea

case 2: {

std::string nombreLinea;

std::cout << "Ingrese el nombre de la línea a eliminar: ";

std::cin >> nombreLinea;

Linea\* linea = nullptr;

Linea\* temp = redMetro.getCabezaLineas();

while (temp != nullptr) {

if (temp->getNombre() == nombreLinea) {

linea = temp;

break;

}

temp = temp->getSiguienteLinea();

}

if (linea == nullptr) {

std::cout << "No se encontró la línea en la red.\n";

} else {

// Verificar si la línea tiene estación de transferencia

Estacion\* estacion = linea->getCabezaEstaciones();

bool tieneTransferencia = false;

while (estacion != nullptr) {

if (estacion->getEsTransferencia()) {

tieneTransferencia = true;

break;

}

estacion = estacion->getSiguienteEstacion();

}

// Si la línea tiene estación de transferencia, no se puede eliminar

if (tieneTransferencia) {

std::cout << "La línea no se pudo eliminar ya que tiene una estación de transferencia.\n";

} else {

// Intentar eliminar la línea

if (redMetro.eliminarLinea(nombreLinea)) {

std::cout << "Línea eliminada correctamente.\n";

} else {

std::cout << "No se encontró la línea o no se pudo eliminar.\n";

}

}

}

break;

}

// Agregar estación

case 3: {

std::string nombreEstacion, nombreLinea;

int tiempoSiguiente, tiempoAnterior;

bool esTransferencia;

std::cout << "Ingrese el nombre de la estación: ";

std::cin >> nombreEstacion;

std::cout << "Ingrese el tiempo en segundos para llegar a la siguiente estación: ";

std::cin >> tiempoSiguiente;

std::cout << "Ingrese el tiempo en segundos desde la estación anterior: ";

std::cin >> tiempoAnterior;

std::cout << "¿Es una estación de transferencia? (1 para sí, 0 para no): ";

std::cin >> esTransferencia;

std::cout << "Ingrese el nombre de la línea a la que pertenece la estación: ";

std::cin >> nombreLinea;

// Crear la nueva estación

Linea\* linea = nullptr;

Estacion\* estacion = new Estacion(nombreEstacion, tiempoSiguiente, tiempoAnterior, esTransferencia);

// Encontrar la línea a la que pertenece la estación

Linea\* temp = redMetro.getCabezaLineas();

while (temp != nullptr) {

if (temp->getNombre() == nombreLinea) {

linea = temp;

break;

}

temp = temp->getSiguienteLinea();

}

// Si la línea no se encontró, mostrar mensaje y liberar memoria

if (linea == nullptr) {

std::cout << "No se encontró la línea en la red.\n";

delete estacion;

} else {

// Obtener la posición donde se va a agregar la estación

int posicion;

std::cout << "Ingrese la posición donde agregar la estación (0 para agregar al inicio): ";

std::cin >> posicion;

// Agregar la estación a la línea

linea->agregarEstacion(estacion, posicion);

std::cout << "Estación agregada correctamente.\n";

}

break;

}

// Eliminar estación

case 4: {

std::string nombreEstacion, nombreLinea;

std::cout << "Ingrese el nombre de la estación a eliminar: ";

std::cin >> nombreEstacion;

std::cout << "Ingrese el nombre de la línea a la que pertenece la estación: ";

std::cin >> nombreLinea;

// Encontrar la línea a la que pertenece la estación

Linea\* linea = nullptr;

Linea\* temp = redMetro.getCabezaLineas();

while (temp != nullptr) {

if (temp->getNombre() == nombreLinea) {

linea = temp;

break;

}

temp = temp->getSiguienteLinea();

}

// Si la línea no se encontró, mostrar mensaje

if (linea == nullptr) {

std::cout << "No se encontró la línea en la red.\n";

} else {

// Intentar eliminar la estación de la línea

if (linea->eliminarEstacion(nombreEstacion)) {

std::cout << "Estación eliminada correctamente.\n";

} else {

std::cout << "No se pudo eliminar la estación.\n";

}

}

break;

}

// Verificar existencia de estación

case 5: {

std::string nombreEstacion, nombreLinea;

std::cout << "Ingrese el nombre de la estación: ";

std::cin >> nombreEstacion;

std::cout << "Ingrese el nombre de la línea: ";

std::cin >> nombreLinea;

// Encontrar la línea en la red

Linea\* linea = nullptr;

Linea\* temp = redMetro.getCabezaLineas();

while (temp != nullptr) {

if (temp->getNombre() == nombreLinea) {

linea = temp;

break;

}

temp = temp->getSiguienteLinea();

}

// Si la línea no se encontró, mostrar mensaje

if (linea == nullptr) {

std::cout << "No se encontró la línea en la red.\n";

} else {

// Verificar si la estación existe en la línea

if (linea->existeEstacion(nombreEstacion)) {

std::cout << "La estación existe en la línea.\n";

} else {

std::cout << "La estación no existe en la línea.\n";

}

}

break;

}

// Contar estaciones en una línea

case 6: {

std::string nombreLinea;

std::cout << "Ingrese el nombre de la línea: ";

std::cin >> nombreLinea;

// Encontrar la línea en la red

Linea\* linea = nullptr;

Linea\* temp = redMetro.getCabezaLineas();

while (temp != nullptr) {

if (temp->getNombre() == nombreLinea) {

linea = temp;

break;

}

temp = temp->getSiguienteLinea();

}

// Si la línea no se encontró, mostrar mensaje

if (linea == nullptr) {

std::cout << "No se encontró la línea en la red.\n";

} else {

// Contar y mostrar el número de estaciones en la línea

std::cout << "La línea " << nombreLinea << " tiene " << linea->getNumEstaciones() << " estaciones.\n";

}

break;

}

// Contar estaciones en la red

case 7: {

// Contar y mostrar el número total de estaciones en la red

std::cout << "La red de metro tiene " << redMetro.contarEstaciones() << " estaciones.\n";

break;

}

// Calcular tiempo de llegada

case 8: {

std::string estacionInicio, estacionDestino, horaInicio;

std::cout << "Ingrese el nombre de la estación de inicio: ";

std::cin >> estacionInicio;

std::cout << "Ingrese el nombre de la estación de destino: ";

std::cin >> estacionDestino;

std::cout << "Ingrese la hora de inicio (formato HH:MM:SS): ";

std::cin >> horaInicio;

// Calcular y mostrar el tiempo estimado de llegada entre dos estaciones

int tiempoLlegada = redMetro.calcularTiempoLlegada(estacionInicio, estacionDestino, horaInicio);

if (tiempoLlegada == -1) {

std::cout << "Alguna de las estaciones no existe en la red.\n";

} else if (tiempoLlegada == -2) {

std::cout << "Las estaciones no están en la misma línea.\n";

} else {

std::cout << "El tiempo estimado de llegada es: " << tiempoLlegada << " segundos\n";

}

break;

}

// Salir del programa

case 0: {

std::cout << "Saliendo del simulador.\n";

break;

}

// Opción inválida

default: {

std::cout << "Opción no válida. Por favor, ingrese una opción válida.\n";

break;

}

}

} while (opcion != 0);

return 0;

}