ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

ESTRUCTURA DE DATOS

DOCUMENTACIÓN DEL PROYECTO

SISTEMA DE GESTIÓN

DE PARQUEADERO

NRC:

1978

Integrantes:

Andrade Danna.

Pérez Carlos

Samaniego Jefferson

Contenido

Sistema de Gestión de Parqueadero

Descripción del Proyecto:

El **Sistema de Gestión de Parqueadero** es una aplicación que gestiona los espacios disponibles en un parqueadero utilizando programación orientada a objetos en **C++**.
El sistema permite la asignación, liberación y visualización de espacios, así como la gestión de vehículos estacionados. Está diseñado para ser modular y escalable, con diferentes clases responsables de manejar la lógica, validaciones, y la interacción con el usuario.

Estructura del Proyecto:

El proyecto está organizado en módulos para una mejor organización y escalabilidad, lo que incluye las clases que gestionan los nodos, manejo de datos el parqueadero, y las validaciones de entrada .

Clase Nodo<T>

La clase Nodo<T> es una estructura de datos fundamental para la implementación de una lista doblemente enlazada. Esta clase permite almacenar información detallada sobre un vehículo y su propietario, así como enlaces a los nodos anterior y siguiente, lo que facilita la navegación bidireccional dentro de la lista.

Atributos

1. **nombre: T**
    - **Descripción**: Almacena el nombre del propietario del vehículo.
    - **Tipo**: T (tipo genérico).
2. **apellido: T**
    - **Descripción**: Almacena el apellido del propietario.
    - **Tipo**: T (tipo genérico).
3. **cedula: T**
    - **Descripción**: Almacena la cédula del propietario.
    - **Tipo**: T (tipo genérico).
4. **idEspacio: int**
    - **Descripción**: Identificador único del espacio en el parqueadero.
    - **Tipo**: int.
5. **placa: T**
    - **Descripción**: Almacena la placa del vehículo.
    - **Tipo**: T (tipo genérico).

6. **marca: T**
    - **Descripción**: Almacena la marca del vehículo.
    - **Tipo**: T (tipo genérico).
7. **color: T**
    - **Descripción**: Almacena el color del vehículo.
    - **Tipo**: T (tipo genérico).
8. **fechaHora: std::string**
    - **Descripción**: Almacena la fecha y hora de registro.
    - **Tipo**: std::string.
9. *siguiente: Nodo* (puntero al siguiente nodo)*
    - **Descripción**: Puntero al siguiente nodo en la lista.
    - **Tipo**: Nodo*.
10. *anterior: Nodo* (puntero al nodo anterior)*
    - **Descripción**: Puntero al nodo anterior en la lista.
    - **Tipo**: Nodo*.

Métodos

Constructores:

- **Nodo(T _nombre, T _apellido, T _cedula, int _idEspacio, T _placa, T _marca, T _color, std::string _fechaHora)**
    - **Descripción**: Constructor que inicializa los datos del nodo con los valores proporcionados.
    - **Parámetros**:
        - _nombre: Nombre del propietario.
        - _apellido: Apellido del propietario.
        - _cedula: Cédula del propietario.
        - _idEspacio: Identificador del espacio.
        - _placa: Placa del vehículo.
        - _marca: Marca del vehículo.
        - _color: Color del vehículo.
        - _fechaHora: Fecha y hora de registro.

Getters y Setters:

- **void setNombre(T _nombre) / T getNombre()**
    - **Descripción**: Establece/Obtiene el nombre del propietario.
- **void setApellido(T _apellido) / T getApellido()**

    - **Descripción**: Establece/Obtiene el apellido del propietario.
- **void setCedula(T _cedula) / T getCedula()**
    - **Descripción**: Establece/Obtiene la cédula del propietario.
- **void setIdEspacio(int _idEspacio) / int getIdEspacio()**
    - **Descripción**: Establece/Obtiene el identificador del espacio en el parqueadero.
- **void setPlaca(T _placa) / T getPlaca()**
    - **Descripción**: Establece/Obtiene la placa del vehículo.
- **void setMarca(T _marca) / T getMarca()**
    - **Descripción**: Establece/Obtiene la marca del vehículo.
- **void setColor(T _color) / T getColor()**
    - **Descripción**: Establece/Obtiene el color del vehículo.
- **void setFechaHora(const std::string& _fechaHora) / std::string getFechaHora() const**
    - **Descripción**: Establece/Obtiene la fecha y hora de registro del nodo.
- *void setSiguiente(Nodo* _siguiente) / Nodo getSiguiente()**
    - **Descripción**: Establece/Obtiene el puntero al siguiente nodo en la lista.
- *void setAnterior(Nodo* _anterior) / Nodo getAnterior()**
    - **Descripción**: Establece/Obtiene el puntero al nodo anterior en la lista.

Clase Parqueadero

La clase Parqueadero maneja la lógica y la estructura de un parqueadero (estacionamiento), almacenando los vehículos y permitiendo su visualización por consola. Esta clase utiliza una matriz para representar los espacios del parqueadero, donde se indica si un espacio está ocupado o libre.

Atributos

1. *matriz: char**
    - **Descripción**: Matriz que representa los espacios del parqueadero.
    - **Tipo**: char**.
    - **Detalles**: Cada celda de la matriz puede contener 'X' para indicar que el espacio está ocupado, o '-' para indicar que está libre.
2. **filas: int**
    - **Descripción**: Número de filas en el parqueadero.
    - **Tipo**: int.
3. **columnas: int**

- **Descripción**: Número de columnas en el parqueadero.
- **Tipo**: int.

4. **totalEspacios: int**
    - **Descripción**: Total de espacios disponibles en el parqueadero.
    - **Tipo**: int.
5. **radio: int (Opcional)**
    - **Descripción**: Radio en el caso de un parqueadero circular.
    - **Tipo**: int.

Métodos

Constructores y Destructores:

- **Parqueadero(int _filas, int _columnas)**
    - **Descripción**: Constructor que inicializa la matriz con las filas y columnas especificadas.
    - **Parámetros**:
        - _filas: Número de filas en el parqueadero.
        - _columnas: Número de columnas en el parqueadero.
    - **Implementación**: Inicializa la matriz con todos los espacios vacíos ('-').
- **~Parqueadero()**
    - **Descripción**: Destructor que libera la memoria de la matriz al destruir el objeto.
    - **Implementación**: Libera la memoria reservada para la matriz.

Gestión del Parqueadero:

- **void mostrarParqueadero()**
    - **Descripción**: Muestra el estado actual del parqueadero, visualizando los espacios ocupados y libres.
    - **Implementación**: Recorre la matriz y muestra cada espacio con su estado ('X' o '-').
- **void ocuparEspacio(int idEspacio)**
    - **Descripción**: Marca un espacio como ocupado.
    - **Parámetros**:
        - idEspacio: Identificador del espacio a ocupar.
    - **Implementación**: Calcula la fila y columna correspondientes y marca el espacio como ocupado ('X').
- **void liberarEspacio(int idEspacio)**

    - **Descripción**: Libera un espacio ocupado.
    - **Parámetros**:
        - idEspacio: Identificador del espacio a liberar.
    - **Implementación**: Calcula la fila y columna correspondientes y marca el espacio como libre ('-').
- **void asignarIdEspacios()**
    - **Descripción**: (Opcional) Asigna un ID único a cada espacio en el parqueadero.
    - **Implementación**: Recorre la matriz asignando un ID único a cada espacio (no implementado en el ejemplo proporcionado).

Clase Validaciones<T>

La clase Validaciones<T> es una clase genérica encargada de validar los datos de entrada, como números, cadenas y otros tipos de datos. Permite asegurarse de que los datos ingresados cumplan con ciertos criterios antes de ser procesados.

Métodos

Constructor:

- **Validaciones()**
    - **Descripción**: Constructor vacío para la inicialización de la clase.
    - **Implementación**: No requiere parámetros y simplemente inicializa la clase.

Métodos de Validación:

- *T ingresar(const char* msj, const char tipo)**
    - **Descripción**: Permite ingresar valores y valida el tipo especificado.
    - **Parámetros**:
        - msj: Mensaje que se muestra al usuario para indicar qué dato debe ingresar.
        - tipo: Tipo de dato que se espera validar. Puede ser "entero", "long", "flotante", "double", "string", "email", "char".
    - **Implementación**: Usa un bucle para leer la entrada del usuario carácter por carácter, validando según el tipo especificado.

Validaciones Disponibles:

- **entero**
    - Valida que el valor ingresado sea un número entero.
- **long**

    - Valida que el valor ingresado sea un número largo.
- **flotante o double**
    - Valida que el valor ingresado sea un número decimal.
- **string**
    - Valida que el valor ingresado sea un texto sin caracteres especiales.
- **email**
    - Verifica que el formato del valor ingresado sea un correo electrónico válido.
- **char**
    - Valida que se ingrese un solo carácter.

Clase Lista_Circular_Doble

Este módulo está diseñado para representar una lista circular doblemente enlazada. Cada nodo de la lista contiene información sobre un vehículo en el parqueadero.

Atributos

- **cabeza: Nodo<T>***
    - **Descripción**: Apunta al primer nodo de la lista circular.

Métodos

Constructor:

- **Lista_Circular_Doble()**
    - **Descripción**: Inicializa una lista vacía, configurando la cabeza como nullptr.

Métodos de Inserción:

- **void InsertarPorCabeza(T nombre, T apellido, T cedula, int idEspacio, T placa, T marca, T color, const std::string& fechaHora)**
    - **Descripción**: Inserta un nodo al inicio de la lista.
    - **Parámetros**:
        - nombre, apellido, cedula, idEspacio, placa, marca, color, fechaHora.
- **void InsertarPorCola(T nombre, T apellido, T cedula, int idEspacio, T placa, T marca, T color, const std::string& fechaHora)**
    - **Descripción**: Inserta un nodo al final de la lista.
    - **Parámetros**:

        - nombre, apellido, cedula, idEspacio, placa, marca, color, fechaHora.

Métodos de Recorrido:

- **void Mostrar()**
    - **Descripción**: Muestra todos los nodos de la lista, recorriéndola de forma circular.

Búsqueda y Eliminación:

- **void BuscarPorPlaca(T placa)**
    - **Descripción**: Busca un nodo por la placa.
    - **Parámetros**: placa.
- **void EliminarPorPlaca(T placa)**
    - **Descripción**: Elimina un nodo por su placa.
    - **Parámetros**: placa.

Métodos de Inicialización y Actualización:

- **void InicializarLista(int totalEspacios)**
    - **Descripción**: Inicializa la lista con nodos vacíos.
    - **Parámetros**: totalEspacios.
- **void ActualizarEspacio(T nombre, T apellido, T cedula, int idEspacio, T placa, T marca, T color, const std::string& fechaHora)**
    - **Descripción**: Permite actualizar los datos de un nodo en la lista.
    - **Parámetros**:
        - nombre, apellido, cedula, idEspacio, placa, marca, color, fechaHora.

Métodos Adicionales:

- **void cifrarCampoDeNodo(int idNodo, const std::string& campo, int desplazamiento)**
    - **Descripción**: Cifra un campo específico en el nodo con el ID proporcionado.
    - **Parámetros**: idNodo, campo, desplazamiento.
- **void cifrarCampoEnTodos(int idNodo, int desplazamiento)**
    - **Descripción**: Cifra varios campos en todos los nodos.
    - **Parámetros**: idNodo, desplazamiento.
- **std::string cifrarString(const std::string& texto, int desplazamiento)**

    - **Descripción**: Aplica un cifrado a la cadena utilizando un desplazamiento.
    - **Parámetros**: texto, desplazamiento.
- **void BuscarPorRangoDeHora(const std::string& horaInicio, const std::string& horaFin)**
    - **Descripción**: Busca nodos cuya hora de entrada esté dentro de un rango especificado.
    - **Parámetros**: horaInicio, horaFin.
- *Nodo<T>\ getCabeza()**
    - **Descripción**: Devuelve el puntero a la cabeza de la lista.

Clase Utils

La clase Utils proporciona un conjunto de funciones utilitarias generales que se utilizan para validar datos y mostrar menús en el sistema del parqueadero. Esta clase incluye métodos para validar identificaciones, correos electrónicos, modificar caracteres en cadenas, validar placas de vehículos, mostrar menús en la consola y verificar la unicidad de campos en una lista circular doblemente enlazada.

Métodos

Validación de Datos:

- **bool validateId(const std::string& idInput)**
    - **Descripción**: Valida una identificación (ID) ecuatoriana de 10 dígitos.
    - **Parámetros**:
        - idInput: Cadena que representa la identificación a validar.
    - **Implementación**: Utiliza el algoritmo de Luhn modificado para verificar la validez del ID.
- **bool validateEmail(const std::string& emailInput)**
    - **Descripción**: Valida una dirección de correo electrónico usando una expresión regular.
    - **Parámetros**:
        - emailInput: Cadena que representa el correo electrónico a validar.
    - **Implementación**: Usa una expresión regular para verificar que el correo electrónico tiene un formato válido.
- **bool validarPlacaEcuador(const std::string& placa)**
    - **Descripción**: Valida una placa de vehículo ecuatoriana.
    - **Parámetros**:

        - placa: Cadena que representa la placa a validar.
    - **Implementación**: Usa expresiones regulares para verificar que la placa cumple con los formatos permitidos en Ecuador.

Modificación de Cadenas:

- **void modifyCharInString(std::string& str, char target, int offset)**
    - **Descripción**: Modifica los caracteres en una cadena desplazando su valor ASCII sí coinciden con el carácter objetivo.
    - **Parámetros**:
        - str: Cadena en la que se realizará la modificación.
        - target: Carácter que será modificado.
        - offset: Desplazamiento a aplicar al carácter.

Gestión de Menús:

- **void mostrarMenu(const std::string opciones[], int numOpciones, int seleccion)**
    - **Descripción**: Muestra un menú en la consola y resalta la opción seleccionada.
    - **Parámetros**:
        - opciones: Arreglo de cadenas con las opciones del menú.
        - numOpciones: Número de opciones en el menú.
        - seleccion: Índice de la opción seleccionada.
- **void mostrarMenuManual(const std::string opciones[], int numOpciones, int seleccion)**
    - **Descripción**: Muestra un menú de manual de usuario en la consola y resalta la opción seleccionada.
    - **Parámetros**:
        - opciones: Arreglo de cadenas con las opciones del manual.
        - numOpciones: Número de opciones en el manual.
        - seleccion: Índice de la opción seleccionada.
- **void mostrarMenuCifrar(const std::string opciones[], int numOpciones, int seleccion)**
    - **Descripción**: Muestra un menú para el cifrado en la consola y resalta la opción seleccionada.
    - **Parámetros**:
        - opciones: Arreglo de cadenas con las opciones del cifrado.

        - numOpciones: Número de opciones en el cifrado.
        - seleccion: Índice de la opción seleccionada.
- **void mostrarMenuCampoCifrar(const std::string opciones[], int numOpciones, int seleccion)**
    - **Descripción**: Muestra un menú para el cifrado por campo en la consola y resalta la opción seleccionada.
    - **Parámetros**:
        - opciones: Arreglo de cadenas con las opciones de cifrado por campo.
        - numOpciones: Número de opciones en el cifrado por campo.
        - seleccion: Índice de la opción seleccionada.

Validación de Campos en Lista:

- *bool isUniqueField(Lista_Circular_Doble<std::string>* lista, const std::string& campo, const std::string& tipo, int id)*
    - **Descripción**: Verifica si un campo es único en una lista circular doblemente enlazada.
    - **Parámetros**:
        - lista: Puntero a la lista circular doble.
        - campo: Valor del campo a verificar.
        - tipo: Tipo de campo a verificar ("cedula", "placa", "id").
        - id: Identificador del espacio a verificar.
- *bool isSpaceOccupied(Lista_Circular_Doble<std::string>* lista, int idEspacio)*
    - **Descripción**: Verifica si un espacio en la lista está ocupado.
    - **Parámetros**:
        - lista: Puntero a la lista circular doble.
        - idEspacio: Identificador del espacio a verificar.

Gestión de Tiempo:

- **std::string getCurrentDateTime()**
    - **Descripción**: Obtiene la fecha y hora actual del sistema en formato de cadena.
    - **Implementación**: Utiliza las bibliotecas estándar de C++ para obtener la hora actual y convertirla a una cadena.

Flujo del Programa Principal (main)

El programa principal (archivo main.cpp) integra todas las clases desarrolladas para gestionar el sistema de parqueadero, permitiendo al usuario realizar operaciones como insertar, eliminar, buscar vehículos y mostrar el estado del parqueadero.

Características

1. **Interacción con el Usuario:**
    - **Menú Interactivo:** Permite manejar el sistema a través de un menú que incluye las siguientes opciones:
        - Insertar vehículo: Solicita los datos del vehículo y lo asigna a un espacio.
        - Eliminar vehículo: Elimina un vehículo de un espacio específico.
        - Mostrar vehículos por hora: Permite buscar vehículos que ingresaron en un rango de horas específico.
        - Manual de usuario: Proporciona instrucciones detalladas sobre cómo usar el sistema.
        - Cifrar: Permite cifrar los datos almacenados para mayor seguridad.
        - Salir: Finaliza el programa.
2. **Navegación por el Menú:**
    - El programa permite navegar a través de las opciones del menú utilizando las flechas del teclado, lo que mejora la experiencia del usuario.

Flujos de Trabajo

1. **Inserción de Vehículo:**
    - Solicitar datos del vehículo (nombre, apellido, cédula, placa, marca, color).
    - Validar la disponibilidad del espacio y la unicidad de la cédula y placa.
    - Insertar el vehículo y actualizar el estado visual del parqueadero.
2. **Eliminación de Vehículo:**
    - Solicitar la placa del vehículo.
    - Verificar si el espacio está ocupado.
    - Eliminar el vehículo y liberar el espacio.
3. **Búsqueda de Vehículos por Hora:**
    - Solicitar la hora de inicio y fin.

    - Mostrar los vehículos estacionados en el rango de tiempo especificado.
4. **Manual de Usuario:**
    - Proporciona instrucciones detalladas sobre cómo utilizar el sistema, incluyendo ejemplos de ingreso de datos y eliminación de vehículos.
5. **Cifrado de Datos:**
    - Permite cifrar los datos de los vehículos para mayor seguridad.
    - Opciones para cifrar todos los datos de un espacio específico o un campo específico de un vehículo.

Herramientas y Tecnologías Utilizadas:

- **Lenguaje**: C++
- **Paradigma**: Programación orientada a objetos.
- **Estructura de Datos**: Lista Circular Doblemente Enlazada.
- **Entrada/Salida**: Operaciones de archivos para guardar y cargar datos.

El sistema de parqueadero proporciona una solución integral para la gestión y administración de vehículos en un estacionamiento, utilizando técnicas avanzadas de programación orientada a objetos y estructuras de datos eficientes.