MANUAL TÉCNICO

Nombre: Josué Nabí Hurtarte Pinto ---------- Carné: 202202481

Descripción general del proyecto:
Se desarrollo un sistema de gestión de aeropuerto utilizando C++. El objetivo principal es aplicar los conocimientos sobre estructuras de datos lineales y el uso de memoria dinámica mediante apuntadores. Se gestionaron vuelos, pasajeros y equipajes usando listas enlazadas, listas circulares, pilas y colas. La aplicación, ejecutada desde una consola, permitirá la simulación de la llegada de aviones y pasajeros, así como la generación de reportes usando Graphviz. Los aviones se almacenarán en listas circulares dobles, mientras que los pasajeros y su equipaje se manejarán con colas y pilas, respectivamente.

Requerimientos mínimos del entorno de desarrollo

- IDE (Entorno de Desarrollo Integrado) en nuestro caso usamos el editor de texto Visual Studio Code.
- C++ (Instalando las extensiones en Visual Studio Code).
- Libreria para lectura de json [Link de Repositorio](https://github.com/open-source-parsers/jsoncpp).
- Graphviz
- Git: Es muy recomendable un control de versiones para gestionar y no perder cambios.

CLASES Y SUS METODOS

Clase Avion

- Constructor: Inicializa un objeto Avion con todos sus atributos dados, incluyendo el número de vuelo, número de registro, modelo, fabricante, año de fabricación, capacidad de pasajeros, peso máximo de despegue, aerolínea y estado del avión.

- Métodos Getters: Proporciona métodos públicos para obtener los valores de cada atributo privado de la clase, tales como el número de vuelo, número de registro, modelo, fabricante, año de fabricación, capacidad, peso máximo de despegue, aerolínea y estado.

- Destructor: Define un destructor para la clase Avion, aunque en este caso no * realiza ninguna acción específica ya que no hay recursos dinámicos que necesiten liberarse explícitamente.

- Atributos privados: Almacena datos específicos de cada avión como variables privadas, incluyendo el número de vuelo, número de registro, modelo, fabricante, año de fabricación, capacidad de pasajeros, peso máximo de despegue, aerolínea y estado.

Clase Pasajero

- Constructor: Inicializa un objeto Pasajero con todos sus atributos dados, incluyendo el nombre, nacionalidad, número de pasaporte, número de vuelo, número de asiento, destino, origen y cantidad de equipaje facturado.

- Métodos Getters: Proporciona métodos públicos para obtener los valores de cada atributo privado de la clase, tales como el nombre, nacionalidad, número de pasaporte, número de vuelo, número de asiento, destino, origen y cantidad de equipaje facturado.

- Destructor: Define un destructor para la clase Pasajero, aunque en este caso no realiza ninguna acción específica ya que no hay recursos dinámicos que necesiten liberarse explícitamente.

- Atributos privados: Almacena datos específicos de cada pasajero como variables privadas, incluyendo el nombre, nacionalidad, número de pasaporte, número de vuelo, número de asiento, destino, origen y cantidad de equipaje facturado.

MAIN

- menu(): Esta función imprime un menú en la consola con varias opciones para el usuario, como carga de aviones, carga de pasajeros, etc. Espera la entrada del usuario para seleccionar una opción.

- cargar_aviones(): Lee datos de un archivo JSON llamado "aviones.json", crea objetos Avion con estos datos y los inserta en una lista circular doble dependiendo de su estado (Disponible o Mantenimiento).

- cargar_pasajeros(): Lee datos de un archivo JSON llamado "pasajeros.json", crea objetos Pasajero con estos datos y los encola en una estructura tipo Cola (colaPasajeros).

- cargar_mov(): Lee movimientos desde un archivo de texto "movimientos.txt". Los movimientos pueden ser ingreso de equipajes a una pila (equipajePasajeros) y cambios de estado de aviones entre Disponible y Mantenimiento en las listas dobles circulares.

- consulta(): Permite al usuario buscar un pasajero por número de pasaporte en una lista doblemente enlazada de pasajeros (listaEquipaje). Si encuentra al pasajero, muestra sus datos.

- reportes(): Presenta un menú de opciones para generar diferentes tipos de reportes, como lista de aviones disponibles, cola de registro de pasajeros, pila de equipaje y lista doble de pasajeros. Cada opción llama a una función específica para generar y guardar un reporte en formato DOT.

- reporte_aviones(): Genera un reporte visual de los aviones disponibles y en mantenimiento utilizando Graphviz. Crea archivos DOT y PNG que muestran los aviones como nodos con conexiones entre ellos según su estado.

- reporte_pila_equipaje(): Genera un reporte visual de la pila de equipaje utilizando Graphviz. Crea un archivo DOT y PNG que muestra cada elemento de la pila como un nodo y las conexiones entre ellos.

- reporte_lista_doble_pasajeros(): Genera un reporte visual de la lista doble de pasajeros utilizando Graphviz. Crea un archivo DOT y PNG que muestra cada pasajero como un nodo y las conexiones entre ellos en la lista.

LISTA DOBLE CIRCULAR

- Constructor y Destructor: El constructor inicializa una lista circular doblemente enlazada vacía. El Destructor libera la memoria ocupada por todos los nodos en la lista, llamando al método clear().

- insert: Inserta un nuevo nodo con el valor value al final de la lista. Si la lista está vacía, el nuevo nodo se convierte en el primer y último nodo. Si no está vacía, el nuevo nodo se inserta al final y los punteros next y prev se actualizan adecuadamente para mantener la estructura circular.

- remove: Elimina el primer nodo que contiene el valor value. Si el nodo a eliminar es el único nodo en la lista, se actualizan first y last a nullptr. Si es el primer nodo, se actualiza first al siguiente nodo. Si es el último nodo, se actualiza last al nodo anterior. Si está en el medio, los punteros next y prev de los nodos vecinos se actualizan adecuadamente.

- getSize: Devuelve el número de elementos en la lista.

- getElement: Devuelve el valor del nodo en la posición especificada por indice. Si el índice está fuera de rango, lanza una excepción.

- isEmpty: Verifica si la lista está vacía.

- Clear: Elimina todos los nodos de la lista y libera la memoria correspondiente, dejando la lista vacía.

- print: Imprime los valores de todos los nodos en la lista en orden, Si la lista está vacía, imprime un mensaje indicando que la lista está vacía.

- buscarPorNumeroDeRegistro: Busca un nodo por su número de registro y devuelve un puntero al nodo que contiene el número de registro especificado.

- eliminarPorNumeroDeRegistro: Elimina el nodo que contiene el número de registro especificado.

LISTA DOBLE ENLAZADA

- Constructor y Destructor: El constructor inicializa una lista doblemente enlazada vacía, con head y tail apuntando a nullptr. El destructor Libera la memoria ocupada por todos los nodos en la lista.

- prepend: Inserta un nuevo nodo con el valor data al inicio de la lista.

- append: Inserta un nuevo nodo con el valor data al final de la lista.

- getSize: Devuelve el número de elementos en la lista.

- getElement: Devuelve el valor del nodo en la posición especificada por pos. Lanza una excepción si la posición está fuera de rango.

- eliminarDelFinal: Elimina el último nodo de la lista.

- eliminarDelInicio: Elimina el primer nodo de la lista.

- buscarPorNumeroDePasaporte: Busca un nodo por su número de pasaporte y devuelve el valor del nodo que contiene el número de pasaporte especificado. Lanza una excepción si no se encuentra.

- ordenarPorNumeroDeAsiento: Ordena los nodos de la lista en orden ascendente según el número de asiento.

- ordenarPorNumeroDeVuelo: Ordena los nodos de la lista en orden ascendente según el número de vuelo.

PILA

- Constructor y Destructor: El constructor inicializa una pila vacía, con top apuntando a nullptr. El destructor libera la memoria ocupada por todos los nodos en la pila llamando al método clear().

- push: Inserta un nuevo nodo con el valor valor en la parte superior de la pila.

- pop: Elimina el nodo en la parte superior de la pila. Muestra un mensaje de error si la pila está vacía.

- topFila: Devuelve el valor del nodo en la parte superior de la pila. Lanza una excepción si la pila está vacía.

- isEmpty: Verifica si la pila está vacía. Devuelve true si top es nullptr, de lo contrario, devuelve false.

- Clear: Elimina todos los nodos de la pila, dejándola vacía.

- getSize: Devuelve el número de elementos en la pila.

- getElement: Devuelve el valor del nodo en la posición especificada por posicion dentro de la pila. Muestra un mensaje de error si la posición está fuera de rango o es negativa.

COLA

- Constructor y Destructor: El constructor inicializa una cola vacía con front y rear apuntando a nullptr y size establecido en 0. El destructor Libera la memoria ocupada por todos los nodos en la cola, llamando al método dequeue() repetidamente hasta que la cola esté vacía.

- isEmpty: Verifica si la cola está vacía.

- getSize: Devuelve el número de elementos en la cola.

- enqueue: Inserta un nuevo nodo con el valor value en la parte trasera (final) de la cola.

- dequeue: Elimina el nodo en la parte delantera (inicio) de la cola y devuelve su valor. Lanza una excepción si la cola está vacía.

- getElement: Devuelve el valor del nodo en la posición especificada por pos dentro de la cola. Lanza una excepción si la posición está fuera de rango.

- getFront: Devuelve el valor del nodo en la parte delantera (inicio) de la cola sin eliminarlo. Lanza una excepción si la cola está vacía.