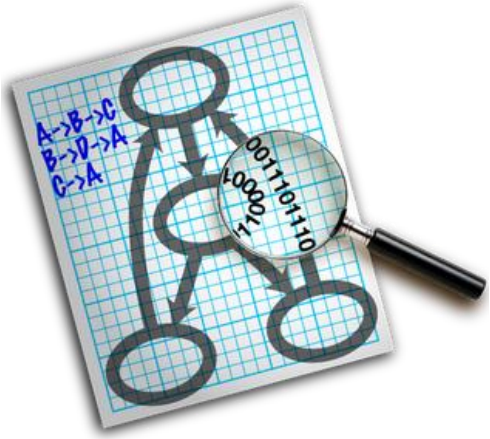
MANUAL TECNICO

Requerimientos del Sistema

UBUNTU

CLION



GRAPHVIZ

Librerias Implementadas

- **<nlohmann/json.hpp>**: Biblioteca para trabajar con datos en formato JSON en C++.

Como instalarlo en Ubuntu

1. Descargar json.hpp desde GitHub:

   wget https://github.com/nlohmann/json/releases/download/v3.10.4/json.hpp

2. Mover json.hpp a tu proyecto:

   mv json.hpp /home/usuario/proyecto/

3. Incluir json.hpp en tu código C++:

   #include "json.hpp"

   using json = nlohmann::json;

- **<iostream>**: Para entrada y salida estándar en consola en C++.
- **<cstdlib>**: Funciones básicas en C++ como gestión de memoria y control de flujo.
- **<fstream>**: Lectura y escritura de archivos en C++.
- **<limits>**: Definición de límites de tipos de datos estándar en C++.
- **<sstream>**: Manipulación avanzada de cadenas de caracteres en C++.

Todas las demás librerías ya vienen el c++ por lo que no es necesario instalarlas.

IDE utilizado

CLion: Es un entorno de desarrollo integrado (IDE) creado por JetBrains, diseñado específicamente para el desarrollo de aplicaciones en C++ y otros lenguajes como C, Python, y JavaScript. Proporciona herramientas avanzadas de edición de código, depuración, refactorización y gestión de proyectos, todo integrado en una interfaz de usuario unificada.

Graficador

Graphviz: Graphviz es un conjunto de herramientas para crear gráficos automáticamente a partir de descripciones en texto, usando un lenguaje llamado DOT. Se usa ampliamente para visualizar datos y estructuras complejas en diversas aplicaciones.

Sistema Operativo

Ubuntu: Ubuntu es un sistema operativo basado en Linux y distribuido de manera gratuita y de código abierto. Es uno de los sistemas operativos más populares y ampliamente utilizados dentro del ecosistema de software libre y de código abierto (FOSS).

Lenguaje Usado

C++: es un lenguaje de programación de propósito general que fue desarrollado a partir del lenguaje C en la década de 1980. Es conocido por su eficiencia, flexibilidad y capacidad para manejar sistemas complejos y aplicaciones de alto rendimiento.

En este caso Ubuntu ya contiene lo que es C++ por lo que solo es de instalarlo por medio de la terminal de Ubuntu

1. Actualizar el índice de paquetes:

   sudo apt update

2. Instalar el compilador g++ y las herramientas de desarrollo:

   sudo apt install g++

3. Verificar la instalación de g++:

   g++ --version

Estructuras Utilizadas en el proyecto

Pila.h

```cpp
#include "Pila.h"
#include <iostream>
#include <cstdlib>
#include <fstream>

Pila::Pila() : tope(nullptr) {}

Pila::~Pila() {
    while (!estaVacia()) {
        desapilar();
    }
}

void Pila::apilar(const Pasajero& pasajero) {
    NodoPasajero* nuevoNodo = new NodoPasajero(pasajero);
    nuevoNodo->siguiente = tope;
    tope = nuevoNodo;
}

void Pila::mostrarPila() const {
    if (estaVacia()) {
        std::cout << "Pila de equipaje está vacía." << std::endl;
        return;
    }

    // Imprimir la pila en consola
    std::cout << "Pila de equipaje:" << std::endl;
    NodoPasajero* actual = tope;
    while (actual) {
        std::cout << "Nombre: " << actual->pasajero.nombre
                  << " - Número de pasaporte: " << actual->pasajero.numeroPasaporte
                  << " - Asiento: " << actual->pasajero.asiento
                  << " - Equipaje facturado: " << (actual->pasajero.equipajeFacturado ? "Si" : "No") << std::endl;
        actual = actual->siguiente;
    }
```

La clase implementa una estructura de datos tipo pila, que sigue el principio primero en entrar ultimo en salir. Permite agregar elementos al tope (apilar), quitar el elemento del tope (desapilar), verificar si está vacía (estaVacia), y mostrar los elementos. También genera un archivo DOT para visualizar la pila como un grafo, convertido luego a PDF con Graphviz.

ListaDoblementeEnlazada.cpp

```cpp
// ListaDoblementeEnlazada.cpp

#include "ListaDoblementeEnlazada.h"
#include <fstream>
#include <iostream>
#include <cstdlib>

ListaDoblementeEnlazada::ListaDoblementeEnlazada() : primero(nullptr), ultimo(nullptr) {
}

ListaDoblementeEnlazada::~ListaDoblementeEnlazada() {
    NodoLista* actual = primero;
    while (actual) {
        NodoLista* siguiente = actual->siguiente;
        delete actual;
        actual = siguiente;
    }
}

void ListaDoblementeEnlazada::insertarOrdenado(const Pasajero& pasajero, const ComparadorPasajero& comparador) {
    NodoLista* nuevoNodo = new NodoLista(pasajero);

    if (!primero) {
        primero = ultimo = nuevoNodo;
    } else {
        NodoLista* actual = primero;
        while (actual) {
            if (comparador(nuevoNodo->pasajero, actual->pasajero)) {
                nuevoNodo->siguiente = actual;
                nuevoNodo->anterior = actual->anterior;
                if (actual->anterior) {
                    actual->anterior->siguiente = nuevoNodo;
                } else {
                    primero = nuevoNodo;
```

La clase ListaDoblementeEnlazada permite gestionar información de pasajeros de manera ordenada y eficiente, facilitando la inserción ordenada, la visualización en orden específico, la búsqueda por número de pasaporte, la eliminación de nodos específicos y la gestión correcta de la memoria.

ListaDoblementeCircular.cpp

```cpp

void ListaAviones::insertarAvionDisponible(const Avion& avion) {
    NodoAvion* nuevoNodo = new NodoAvion(new Avion(avion));
    if (!avionesDisponibles) {
        nuevoNodo->siguiente = nuevoNodo;
        nuevoNodo->anterior = nuevoNodo;
        avionesDisponibles = nuevoNodo;
    } else {
        NodoAvion* ultimoNodo = avionesDisponibles->anterior;
        nuevoNodo->siguiente = avionesDisponibles;
        nuevoNodo->anterior = ultimoNodo;
        avionesDisponibles->anterior = nuevoNodo;
        ultimoNodo->siguiente = nuevoNodo;
    }
}

void ListaAviones::insertarAvionMantenimiento(const Avion& avion) {
    NodoAvion* nuevoNodo = new NodoAvion(new Avion(avion));
    if (!avionesMantenimiento) {
        nuevoNodo->siguiente = nuevoNodo;
        nuevoNodo->anterior = nuevoNodo;
        avionesMantenimiento = nuevoNodo;
    } else {
        NodoAvion* ultimoNodo = avionesMantenimiento->anterior;
        nuevoNodo->siguiente = avionesMantenimiento;
        nuevoNodo->anterior = ultimoNodo;
        avionesMantenimiento->anterior = nuevoNodo;
        ultimoNodo->siguiente = nuevoNodo;
    }
```

Las listas doblemente circulares permiten almacenar datos de forma circular, donde cada nodo tiene referencias tanto al siguiente como al anterior. Esto facilita agregar elementos al principio o al final de la lista y permite un acceso eficiente desde cualquier nodo. En el contexto de las funciones proporcionadas: insertarAvionDisponible: Agrega un avión

disponible al final de una lista circular. insertarAvionMantenimiento: Inserta un avión en una lista circular destinada para aviones en mantenimiento.

CMakeLists.txt

El archivo CMakeLists.txt se utiliza para configurar y construir un proyecto de C++ llamado "EDDFINAL", estableciendo el estándar de C++ (C++17), agregando archivos fuente al ejecutable y vinculando bibliotecas necesarias para funcionalidades específicas, como en este caso, la visualización de gráficos con Graphviz.

```
cmake_minimum_required(VERSION 3.28)
project(EDDFINAL)

set(CMAKE_CXX_STANDARD 17)

# Añadir el ejecutable
add_executable(EDDFINAL
        main.cpp
        GestorAviones.h
        Avion.h
        GestorAviones.cpp
        GestorPasajeros.h
        GestorPasajeros.cpp
        NodoAvion.h
        ListaAviones.h
        ListaAviones.cpp
        NodoPasajero.h
        Pasajero.h
        ListaDoblementeEnlazada.cpp
        Pila.cpp
        Pila.h
        ListaDoblementeEnlazada.h
        Pasajero.cpp
)
target_link_libraries(EDDFINAL PUBLIC cgraph gvc)
```