# MANUAL TECNICO

## INTRODUCCIÓN

Este manual técnico describe la implementación de una simulación de un aeropuerto, donde se gestiona la llegada y el estado de los aviones, así como el manejo de pasajeros y su equipaje. El objetivo principal de esta simulación es proporcionar una herramienta que permita al usuario interactuar con diferentes aspectos operativos del aeropuerto, desde el control de los estados de los aviones hasta la administración del flujo de pasajeros y su equipaje.

## CONTEXTO Y OBJETIVOS

En el entorno aeroportuario, la gestión eficiente de los recursos es crucial para asegurar la fluidez y la puntualidad de las operaciones. Este sistema está diseñado para simular un escenario donde los aviones se clasifican y gestionan según su disponibilidad o necesidad de mantenimiento, y los pasajeros son procesados a medida que registran su equipaje y se preparan para embarcar.

## ESTRUCTURAS DE DATOS

Para simular de manera eficiente y ordenada los procesos del aeropuerto, se emplean diversas estructuras de datos:

**Listas Circulares Dobles:** Utilizadas para mantener dos listados de aviones, uno para los aviones disponibles y otro para los aviones en mantenimiento. Estas listas permiten un acceso rápido y eficiente, así como una fácil manipulación de los datos.

**Pila:** Empleada para manejar el equipaje de los pasajeros. Al procesar a los pasajeros en la cola de registro, aquellos con equipaje son apilados para un seguimiento fácil y ordenado.

**Lista Enlazada Doble:** Utilizada para almacenar y ordenar a los pasajeros que han completado su registro. Esta lista facilita la ordenación y consulta de los pasajeros según su número de vuelo y asiento.

**Cola:** utilizada para almacenar a los pasajeros en la secuencia en la que llegan a su registro. La estructura de cola asegura que los pasajeros se procesen en el orden de llegada, siguiendo el principio de primero en entrar, primero en salir.

## Código de Listas Circulares Dobles con constructores usados:

```cpp
#ifndef LISTADOBLECIRCULAR_H
#define LISTADOBLECIRCULAR_H

#include "Nodo_avion.h"
#include <iostream>

class ListaDobleCircular {
private:
    Nodo_avion* cabeza;

public:
    ListaDobleCircular() : cabeza(nullptr) {}

    ~ListaDobleCircular() {
        if (!cabeza) return;

        Nodo_avion* actual = cabeza;
        do {
            Nodo_avion* siguiente = actual->siguiente;
            delete actual;
            actual = siguiente;
        } while (actual != cabeza);
    }
```

```cpp
    // Metodo para agregar aviones a la lista
    void agregarAvion(Avion a) {
        Nodo_avion* nuevo = new Nodo_avion(a);

        if (!cabeza) {
            cabeza = nuevo;
            cabeza->siguiente = cabeza;
            cabeza->anterior = cabeza;
        } else {
            Nodo_avion* ultimo = cabeza->anterior;
            ultimo->siguiente = nuevo;
            nuevo->anterior = ultimo;
            nuevo->siguiente = cabeza;
            cabeza->anterior = nuevo;
        }
    }
```

```cpp
    // Metodo para imprimir la lista de Aviones
    void imprimirLista() {
        if (!cabeza) return;

        Nodo_avion* actual = cabeza;
        do {
            std::cout << "Vuelo: " << actual->avion.vuelo << std::endl;
            std::cout << "Nmero de Registro: " << actual->avion.numero_de_registro << std::endl;
            std::cout << "Modelo: " << actual->avion.modelo << std::endl;
            std::cout << "Fabricante: " << actual->avion.fabricante << std::endl;
            std::cout << "Anio de Fabricaciin: " << actual->avion.ano_fabricacion << std::endl;
            std::cout << "Capacidad: " << actual->avion.capacidad << std::endl;
            std::cout << "Peso Maximo de Despegue: " << actual->avion.peso_max_despegue << std::endl;
            std::cout << "Aerolinea: " << actual->avion.aerolinea << std::endl;
            std::cout << "Estado: " << actual->avion.estado << std::endl;
            std::cout << "-------------------------" << std::endl;
            std::cout << std::endl;
            actual = actual->siguiente;
        } while (actual != cabeza);
    }
```

```cpp
bool estaVacia() const {
    return cabeza == nullptr;
}
```

```cpp
Avion eliminar(const std::string& numero_de_registro) {
    if (!cabeza) return Avion();  // La lista está vacía

    Nodo_avion* actual = cabeza;
    Nodo_avion* previo = nullptr;

    do {
        if (actual->avion.numero_de_registro == numero_de_registro) {
            Avion avionEliminado = actual->avion;

            if (actual->siguiente == actual) {
                // Solo hay un nodo en la lista
                delete actual;
                cabeza = nullptr;
            } else {
                // Hay más de un nodo en la lista
                if (actual == cabeza) {
                    cabeza = actual->siguiente;
                }
                previo = actual->anterior;
                previo->siguiente = actual->siguiente;
                actual->siguiente->anterior = previo;
```

```cpp
                delete actual;
            }

            return avionEliminado;
        }
        actual = actual->siguiente;
    } while (actual != cabeza);

    return Avion();  // No se encontró el avión con el número de registro dado
}
```

```cpp
// Método para generar el archivo DOT para Graphviz y abrirlo
void generarDot(const std::string& nombreArchivo) const {
    std::ofstream archivo(nombreArchivo);
    if (!archivo.is_open()) {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
        return;
    }

    archivo << "digraph G {" << std::endl;
    archivo << "    node [shape=record];" << std::endl;

    if (cabeza) {
        Nodo_avion* actual = cabeza;
        int id = 0;
        do {
            archivo << "    nodo" << id << " [label=\"{|Vuelo: "
                    << actual->avion.vuelo << "\\nNúmero de Registro: "
                    << actual->avion.numero_de_registro << "\\nModelo: "
                    << actual->avion.modelo << "\\nFabricante: "
                    << actual->avion.fabricante << "\\nAño: "
                    << actual->avion.ano_fabricacion << "\\nCapacidad: "
                    << actual->avion.capacidad << "\\nPeso Máximo: "
                    << actual->avion.peso_max_despegue << "\\nAerolínea: "
                    << actual->avion.aerolinea << "\\nEstado: "
                    << actual->avion.estado << "|}\"];" << std::endl;
```

```
        archivo << "}" << std::endl;
        archivo.close();

        // Mensaje de confirmación para verificar que el archivo se ha creado
        std::cout << "Archivo DOT generado: " << nombreArchivo << std::endl;

        // Comando para abrir el archivo DOT con la aplicación predeterminada
        std::string comando = "dot -Tpng " + nombreArchivo + " -o " + nombreArchivo + ".png";
        int result = std::system( Command: comando.c_str());

        // Verificar si el comando se ejecutó correctamente
        if (result == 0) {
            // Abrir la imagen generada
#ifdef _WIN32
            std::string openCommand = "start " + nombreArchivo + ".png";
#elif __APPLE__  #ifdef _WIN32
            std::string openCommand = "open " + nombreArchivo + ".png";
#else  #elif __APPLE__
            std::string openCommand = "xdg-open " + nombreArchivo + ".png";
#endif  #elif __APPLE__ #else
            std::system( Command: openCommand.c_str());
        } else {
            std::cerr << "Error al generar la imagen PNG con Graphviz." << std::endl;
        }
    }
```

```
#ifndef NODO_AVION_H
#define NODO_AVION_H

#include "Avion.h"

struct Nodo_avion {
    Avion avion;
    Nodo_avion* siguiente;
    Nodo_avion* anterior;

    Nodo_avion(Avion a) : avion( a), siguiente(nullptr), anterior(nullptr) {}
};

#endif //NODO_AVION_H
```

```
#ifndef AVION_H
#define AVION_H

#include <string>

struct Avion {
    std::string vuelo;
    std::string numero_de_registro;
    std::string modelo;
    std::string fabricante;
    int ano_fabricacion;
    int capacidad;
    int peso_max_despegue;
    std::string aerolinea;
    std::string estado;
};

#endif //AVION_H
```

## Código de Pila y los constructores usados:

```cpp
#ifndef PILA_H
#define PILA_H

#include "Nodo_equipaje.h"
#include <iostream>

class Pila {
private:
    Nodo_equipaje* tope;

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

    ~Pila() {
        while (!esta_vacia()) {
            desapilar();
        }
    }

    bool esta_vacia() const {
        return tope == nullptr;
    }
```

```cpp
    void apilar(const Equipaje& equipaje) {
        Nodo_equipaje* nuevo_nodo = new Nodo_equipaje(equipaje);
        if (esta_vacia()) {
            tope = nuevo_nodo;
        } else {
            nuevo_nodo->siguiente = tope;
            tope->anterior = nuevo_nodo;
            tope = nuevo_nodo;
        }
    }

    Equipaje desapilar() {
        if (esta_vacia()) {
            std::cerr << "La pila está vacía" << std::endl;
            throw std::runtime_error("Pila vacía");
        } else {
            Nodo_equipaje* nodo_a_eliminar = tope;
            Equipaje equipaje_desapilado = nodo_a_eliminar->equipaje;
            tope = tope->siguiente;
            if (tope != nullptr) {
                tope->anterior = nullptr;
            }
            delete nodo_a_eliminar;
            return equipaje_desapilado;
        }
    }
```

```cpp
    Equipaje ver_tope() const {
        if (esta_vacia()) {
            std::cerr << "La pila está vacía" << std::endl;
            throw std::runtime_error("Pila vacía");
        } else {
            return tope->equipaje;
        }
    }

    void imprimir() const {
        if (esta_vacia()) {
            std::cout << "La pila está vacía" << std::endl;
        } else {
            Nodo_equipaje* actual = tope;
            while (actual != nullptr) {
                std::cout << actual->equipaje.nombre << std::endl;
                std::cout << actual->equipaje.numero_de_pasaporte << std::endl;
                std::cout << actual->equipaje.equipaje << std::endl;
                std::cout << "---------------------------------------- \n";
                actual = actual->siguiente;
            }
        }
    }
```

```cpp
// Método para generar el archivo DOT para Graphviz y abrirlo
void generarDot(const std::string& nombreArchivo) const {
    std::ofstream archivo(nombreArchivo);
    if (!archivo.is_open()) {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
        return;
    }

    archivo << "digraph G {" << std::endl;
    archivo << "    node [shape=record];" << std::endl;

    Nodo_equipaje* actual = tope;
    int id = 0;
    while (actual != nullptr) {
        archivo << "    nodo" << id << " [label=\"{<siguiente> | Nombre: "
                << actual->equipaje.nombre << "\\nPasaporte: "
                << actual->equipaje.numero_de_pasaporte << "\\nEquipaje: "
                << actual->equipaje.equipaje << " | <anterior>}\"];" << std::endl;
        if (actual->siguiente != nullptr) {
            archivo << "    nodo" << id << ":siguiente -> nodo" << id + 1 << ":anterior;" << std::endl;
        }
        actual = actual->siguiente;
        id++;
    }

    archivo << "}" << std::endl;
    archivo.close();
```

```cpp
        archivo << "}" << std::endl;
        archivo.close();

        // Mensaje de confirmación para verificar que el archivo se ha creado
        std::cout << "Archivo DOT generado: " << nombreArchivo << std::endl;

        // Comando para abrir el archivo DOT con la aplicación predeterminada
        std::string comando = "dot -Tpng " + nombreArchivo + " -o " + nombreArchivo + ".png";
        int result = std::system( Command: comando.c_str());

        // Verificar si el comando se ejecutó correctamente
        if (result == 0) {
            // Abrir la imagen generada
#ifdef _WIN32
            std::string openCommand = "start " + nombreArchivo + ".png";
#elif __APPLE__  #ifdef _WIN32
            std::string openCommand = "open " + nombreArchivo + ".png";
#else  #elif __APPLE__
            std::string openCommand = "xdg-open " + nombreArchivo + ".png";
#endif  #elif __APPLE__ #else
            std::system( Command: openCommand.c_str());
        } else {
            std::cerr << "Error al generar la imagen PNG con Graphviz." << std::endl;
        }
    }
};
```

```cpp
#ifndef EQUIPAJE_H
#define EQUIPAJE_H

#include <string>

struct Equipaje{
    std::string nombre;
    std::string numero_de_pasaporte;
    int equipaje;
};

#endif //EQUIPAJE_H
```

```cpp
#ifndef NODO_EQUIPAJE_H
#define NODO_EQUIPAJE_H

#include "Equipaje.h"

struct Nodo_equipaje{
    Equipaje equipaje;
    Nodo_equipaje* siguiente;
    Nodo_equipaje* anterior;

    Nodo_equipaje(Equipaje e): equipaje( e), siguiente(nullptr), anterior(nullptr){}
};

#endif //NODO_EQUIPAJE_H
```

## Código de Lista Enlazada Doble y sus constructores:

```cpp
#ifndef LISTADOBLE_H
#define LISTADOBLE_H

#include "Nodo_atendidos.h"
#include <iostream>

class ListaDoble {
private:
    Nodo_atendidos* cabeza;

public:
    ListaDoble() : cabeza(nullptr) {}

    // Destructor para liberar la memoria de los nodos
    ~ListaDoble() {
        Nodo_atendidos* actual = cabeza;
        while (actual != nullptr) {
            Nodo_atendidos* temp = actual;
            actual = actual->siguiente;
            delete temp;
        }
        cabeza = nullptr;
    }
```

```cpp
    // Método para añadir al final de la lista
    void agregarFinal(Pasajero pa) {
        Nodo_atendidos* nuevo_nodo = new Nodo_atendidos(pa);
        if (cabeza == nullptr) {
            cabeza = nuevo_nodo;
        } else {
            Nodo_atendidos* actual = cabeza;
            while (actual->siguiente != nullptr) {
                actual = actual->siguiente;
            }
            actual->siguiente = nuevo_nodo;
            nuevo_nodo->anterior = actual;
        }
    }
```

```cpp
    // Método para añadir al principio de la lista
    void agregarInicio(Pasajero pa) {
        Nodo_atendidos* nuevo_nodo = new Nodo_atendidos(pa);
        if (cabeza == nullptr) {
            cabeza = nuevo_nodo;
        } else {
            nuevo_nodo->siguiente = cabeza;
            cabeza->anterior = nuevo_nodo;
            cabeza = nuevo_nodo;
        }
    }
```

```cpp
// Método para eliminar un nodo con un dato especifico (Pasajero)
void eliminarNodo(Pasajero pa) {
    Nodo_atendidos* actual = cabeza;
    while (actual != nullptr) {
        if (actual->pasajero.nombre == pa.nombre &&
            actual->pasajero.numero_de_pasaporte == pa.numero_de_pasaporte &&
            actual->pasajero.vuelo == pa.vuelo) {

            if (actual->anterior != nullptr) {
                actual->anterior->siguiente = actual->siguiente;
            }
            if (actual->siguiente != nullptr) {
                actual->siguiente->anterior = actual->anterior;
            }
            if (actual == cabeza) {
                cabeza = actual->siguiente;
            }
            delete actual;
            return;
        }
        actual = actual->siguiente;
    }
}
```

```cpp
// Método para imprimir la lista (imprime datos relevantes de Pasajero)
void imprimirLista() {
    Nodo_atendidos* actual = cabeza;
    while (actual != nullptr) {
        std::cout << "Nombre: " << actual->pasajero.nombre << ", ";
        std::cout << "Pasaporte: " << actual->pasajero.numero_de_pasaporte << ", ";
        std::cout << "Vuelo: " << actual->pasajero.vuelo << std::endl;
        std::cout << "Vuelo: " << actual->pasajero.asiento << std::endl;
        std::cout << "---------------------------------------- \n";
        actual = actual->siguiente;
    }
    std::cout << std::endl;
}
```

```cpp
void ordenarLista() {
    if (cabeza == nullptr || cabeza->siguiente == nullptr) return;

    bool huboIntercambio;
    do {
        huboIntercambio = false;
        Nodo_atendidos* actual = cabeza;
        while (actual->siguiente != nullptr) {
            Nodo_atendidos* siguiente = actual->siguiente;
            if (actual->pasajero.vuelo > siguiente->pasajero.vuelo ||
                (actual->pasajero.vuelo == siguiente->pasajero.vuelo && actual->pasajero.asiento > siguiente->pasajero.asiento)) {
                // Intercambio de datos de los pasajeros
                Pasajero temp = actual->pasajero;
                actual->pasajero = siguiente->pasajero;
                siguiente->pasajero = temp;
                huboIntercambio = true;
                }
            actual = siguiente;
        }
    } while (huboIntercambio);
}
```

```cpp
void generarDot(const std::string& nombreArchivo) {
    std::ofstream archivo(nombreArchivo);
    if (!archivo.is_open()) {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
        return;
    }

    archivo << "digraph G {" << std::endl;
    archivo << "    node [shape=record];" << std::endl;

    Nodo_atendidos* actual = cabeza;
    int id = 0;
    while (actual != nullptr) {
        archivo << "    nodo" << id << " [label=\"{<anterior> | Nombre: "
                << actual->pasajero.nombre << "\\nPasaporte: "
                << actual->pasajero.numero_de_pasaporte << "\\nNacionalidad: "
                << actual->pasajero.nacionalidad << "\\nVuelo: "
                << actual->pasajero.vuelo << "\\nAsiento: "
                << actual->pasajero.asiento << "\\nDestino: "
                << actual->pasajero.destino << "\\nOrigen: "
                << actual->pasajero.origen << "\\nEquipaje Facturado: "
                << actual->pasajero.equipaje_facturado << " | <siguiente>}\"];" << std::endl;
        actual = actual->siguiente;
        id++;
    }
```

```
    actual = cabeza;
    id = 0;
    while (actual != nullptr) {
        if (actual->siguiente != nullptr) {
            archivo << "    nodo" << id << ":siguiente -> nodo" << id + 1 << ":anterior;" << std::endl;
            archivo << "    nodo" << id + 1 << ":anterior -> nodo" << id << ":siguiente;" << std::endl;
        }
        actual = actual->siguiente;
        id++;
    }

    archivo << "}" << std::endl;
    archivo.close();

    // Mensaje de confirmación para verificar que el archivo se ha creado
    std::cout << "Archivo DOT generado: " << nombreArchivo << std::endl;

    // Comando para abrir el archivo DOT con la aplicación predeterminada
    std::string comando = "dot -Tpng " + nombreArchivo + " -o " + nombreArchivo + ".png";
    int result = std::system(Command: comando.c_str());

    // Verificar si el comando se ejecutó correctamente
    if (result == 0) {
        // Abrir la imagen generada
```

```
#def _WIN32
        std::string openCommand = "start " + nombreArchivo + ".png";
#if __APPLE__  #ifdef _WIN32
        std::string openCommand = "open " + nombreArchivo + ".png";
#se  #elif __APPLE__
        std::string openCommand = "xdg-open " + nombreArchivo + ".png";
#dif  #elif __APPLE__ #else
        std::system(Command: openCommand.c_str());
    } else {
        std::cerr << "Error al generar la imagen PNG con Graphviz." << std::endl;
    }
}
```

```
// Método para buscar un pasajero por número de pasaporte
bool buscarPorPasaporte(const std::string& numeroPasaporte) {
    Nodo_atendidos* actual = cabeza;
    while (actual != nullptr) {
        if (actual->pasajero.numero_de_pasaporte == numeroPasaporte) {
            std::cout << "---------------------------------------- \n";
            std::cout << "Nombre: " << actual->pasajero.nombre << std::endl;
            std::cout << "Pasaporte: " << actual->pasajero.numero_de_pasaporte << std::endl;
            std::cout << "Vuelo: " << actual->pasajero.vuelo << std::endl;
            std::cout << "Asiento: " << actual->pasajero.asiento << std::endl;
            std::cout << "Destino: " << actual->pasajero.destino << std::endl;
            std::cout << "Origen: " << actual->pasajero.origen << std::endl;
            std::cout << "Equipaje: " << actual->pasajero.equipaje_facturado << std::endl;
            return true; // Se encontró el pasajero
        }
        actual = actual->siguiente;
    }
    return false; // No se encontró el pasajero
}



#dif //LISTADOBLE_H
```

En este caso se usó los mismos constructores que de Personas ya que se necesitaba la misma información.

## Código de Cola y sus constructores:

```
#ifndef COLA_H
#define COLA_H

#include ...

class Cola {
private:
    Nodo_pasajero* primero;
    Nodo_pasajero* ultimo;

public:
    Cola() : primero(nullptr), ultimo(nullptr) {}

    ~Cola() {
        while (primero != nullptr) {
            Nodo_pasajero* temp = primero;
            primero = primero->siguiente;
            delete temp;
        }
        ultimo = nullptr;
    }

    bool esta_vacia() {
        return primero == nullptr;
    }

    void encolar(Nodo_pasajero valor) {
```

```
    void encolar(Nodo_pasajero valor) {
        Nodo_pasajero* nuevo = new Nodo_pasajero(valor);
        if (esta_vacia()) {
            primero = nuevo;
            ultimo = nuevo;
        } else {
            ultimo->siguiente = nuevo;
            ultimo = nuevo;
        }
    }

    Pasajero desencolar() {
        if (esta_vacia()) {
            std::cerr << "La cola está vacía" << std::endl;
            return Pasajero();
        }
        Nodo_pasajero* temp = primero;
        Pasajero valor = primero->pasajero;
        primero = primero->siguiente;
        delete temp;
        return valor;
    }
```

```
void imprimir() {
    Nodo_pasajero* actual = primero;
    std::cout << "Contenido de la cola:" << std::endl;
    while (actual != nullptr) {
        std::cout << "Nombre: " << actual->pasajero.nombre << std::endl;
        std::cout << "Nacionalidad: " << actual->pasajero.nacionalidad << std::endl;
        std::cout << "Numero de pasaporte: " << actual->pasajero.numero_de_pasaporte << std::endl;
        std::cout << "Vuelo: " << actual->pasajero.vuelo << std::endl;
        std::cout << "Destino: " << actual->pasajero.destino << std::endl;
        std::cout << "Origen: " << actual->pasajero.origen << std::endl;
        std::cout << "Equipaje facturado: " << actual->pasajero.equipaje_facturado << std::endl;
        std::cout << "--------------------------" << std::endl;
        std::cout << std::endl;
        actual = actual->siguiente;
    }
}
```

```
void generarDot(const std::string& nombreArchivo) const {
    std::ofstream archivo(nombreArchivo);
    if (!archivo.is_open()) {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
        return;
    }

    archivo << "digraph G {" << std::endl;
    archivo << "    node [shape=record];" << std::endl;

    Nodo_pasajero* actual = primero;
    int id = 0;
    while (actual != nullptr) {
        archivo << "    nodo" << id << " [label=\"{|Nombre: "
                << actual->pasajero.nombre << "\\nNacionalidad: "
                << actual->pasajero.nacionalidad << "\\nPasaporte: "
                << actual->pasajero.numero_de_pasaporte << "\\nVuelo: "
                << actual->pasajero.vuelo << "\\nDestino: "
                << actual->pasajero.destino << "\\nOrigen: "
                << actual->pasajero.origen << "\\nEquipaje: "
                << actual->pasajero.equipaje_facturado << "|}\"];" << std::endl;
        if (actual->siguiente != nullptr) {
            archivo << "    nodo" << id << " -> nodo" << id + 1 << ";" << std::endl;
        }
        actual = actual->siguiente;
        id++;
    }
```

```
        archivo << "}" << std::endl;
        archivo.close();

        // Mensaje de confirmación para verificar que el archivo se ha creado
        std::cout << "Archivo DOT generado: " << nombreArchivo << std::endl;

        // Comando para abrir el archivo DOT con la aplicación predeterminada
        std::string comando = "dot -Tpng " + nombreArchivo + " -o " + nombreArchivo + ".png";
        int result = std::system( Command: comando.c_str());

        // Verificar si el comando se ejecutó correctamente
        if (result == 0) {
            // Abrir la imagen generada
#ifdef _WIN32
            std::string openCommand = "start " + nombreArchivo + ".png";
#elif __APPLE__...
#else... #elif __APPLE__ #else
            std::system( Command: openCommand.c_str());
        } else {
            std::cerr << "Error al generar la imagen PNG con Graphviz." << std::endl;
        }
    }
```

```
    bool buscar_por_pasaporte(const std::string& numero_de_pasaporte) {
        Nodo_pasajero* actual = primero;
        while (actual != nullptr) {
            if (actual->pasajero.numero_de_pasaporte == numero_de_pasaporte) {
                std::cout << "---------------------------------------- \n";
                std::cout << "Nombre: " << actual->pasajero.nombre << std::endl;
                std::cout << "Pasaporte: " << actual->pasajero.numero_de_pasaporte << std::endl;
                std::cout << "Vuelo: " << actual->pasajero.vuelo << std::endl;
                std::cout << "Asiento: " << actual->pasajero.asiento << std::endl;
                std::cout << "Destino: " << actual->pasajero.destino << std::endl;
                std::cout << "Origen: " << actual->pasajero.origen << std::endl;
                std::cout << "Equipaje: " << actual->pasajero.equipaje_facturado << std::endl;
                return true;
            }
            actual = actual->siguiente;
        }
        return false;
    }

};

#endif //COLA_H
```

```cpp
#ifndef PASAJERO_H
#define PASAJERO_H

#include <string>

struct Pasajero{
    std::string nombre;
    std::string nacionalidad;
    std::string numero_de_pasaporte;
    std::string vuelo;
    int asiento;
    std::string destino;
    std::string origen;
    int equipaje_facturado;
};

#endif //PASAJERO_H
```

```cpp
#ifndef NODO_EQUIPAJE_H
#define NODO_EQUIPAJE_H

#include "Equipaje.h"

struct Nodo_equipaje{
    Equipaje equipaje;
    Nodo_equipaje* siguiente;
    Nodo_equipaje* anterior;

    Nodo_equipaje(Equipaje e): equipaje(e), siguiente(nullptr), anterior(nullptr){}
};

#endif //NODO_EQUIPAJE_H
```