Manual Técnico

Este manual técnico describe la funcionalidad y estructura de un programa C++ diseñado para gestionar un sistema de aviones, pasajeros y movimientos de equipajes. El sistema utiliza diversas estructuras de datos como colas, listas dobles y pilas para organizar y procesar la información.

El programa requiere las siguientes bibliotecas y archivos:

- <iostream>: Entrada y salida estándar.
- <fstream>: Manejo de archivos.
- json.hpp: Biblioteca JSON para C++ (nlohmann/json).
- "cola.h": Definición de la clase Cola.
- "listadoble.h": Definición de la clase Listadoble.
- "pila.h": Definición de la clase Pila.
- "ListaPasajeros.h": Definición de la clase ListaPasajeros

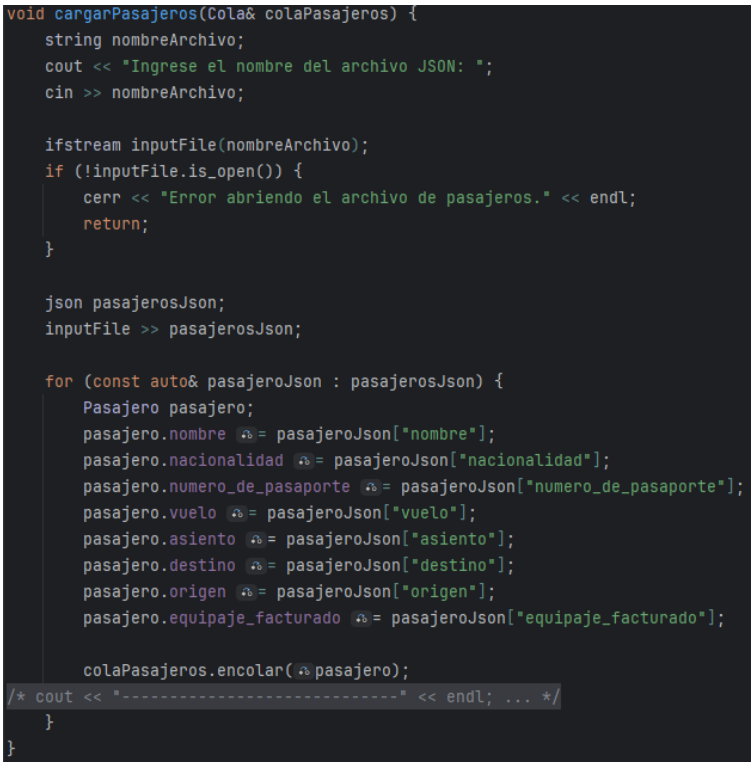
Funciones del Programa

```
void cargarAviones(Listadoble& listaDisponibles, Listadoble& listaMantenimiento) {
    string nombreArchivo;
    cout << "Ingrese el nombre del archivo JSON: ";
    cin >> nombreArchivo;

    ifstream inputFile(nombreArchivo);
    if (!inputFile.is_open()) {
        cerr << "Error abriendo el archivo de aviones." << endl;
        return;
    }

    json avionesJson;
    inputFile >> avionesJson;

    for (const auto& avionJson : avionesJson) {
        Avion avion;
        avion.vuelo = avionJson["vuelo"];
        avion.numero_de_registro = avionJson["numero_de_registro"];
        avion.modelo = avionJson["modelo"];
        avion.fabricante = avionJson["fabricante"];
        avion.ano_fabricacion = avionJson["ano_fabricacion"];
        avion.capacidad = avionJson["capacidad"];
        avion.peso_max_despegue = avionJson["peso_max_despegue"];
        avion.aerolinea = avionJson["aerolinea"];
        avion.estado = avionJson["estado"];

        if (avion.estado == "Disponible") {
            listaDisponibles.insertarAlFinal(avion);
        } else if (avion.estado == "Mantenimiento") {
            listaMantenimiento.insertarAlFinal(avion);
        } else {
            cerr << "Error: Estado desconocido para el avion." << endl;
        }

        //cout << "Avion cargado: " << avion.modelo << ", Estado: " << avion.estado << endl;
    }

}
```

Carga los datos de aviones desde un archivo JSON y los clasifica en listas de disponibles y en mantenimiento.

Funcionamiento:

- Solicita el nombre del archivo JSON.
- Abre el archivo y lee su contenido.
- Itera sobre cada avión y lo clasifica según su estado.



```
void cargarPasajeros(Cola& colaPasajeros) {
    string nombreArchivo;
    cout << "Ingrese el nombre del archivo JSON: ";
    cin >> nombreArchivo;

    ifstream inputFile(nombreArchivo);
    if (!inputFile.is_open()) {
        cerr << "Error abriendo el archivo de pasajeros." << endl;
        return;
    }

    json pasajerosJson;
    inputFile >> pasajerosJson;

    for (const auto& pasajeroJson : pasajerosJson) {
        Pasajero pasajero;
        pasajero.nombre = pasajeroJson["nombre"];
        pasajero.nacionalidad = pasajeroJson["nacionalidad"];
        pasajero.numero_de_pasaporte = pasajeroJson["numero_de_pasaporte"];
        pasajero.vuelo = pasajeroJson["vuelo"];
        pasajero.asiento = pasajeroJson["asiento"];
        pasajero.destino = pasajeroJson["destino"];
        pasajero.origen = pasajeroJson["origen"];
        pasajero.equipaje_facturado = pasajeroJson["equipaje_facturado"];

        colaPasajeros.encolar(pasajero);
/* cout << "----------------------------" << endl; ... */
    }
}
```

Carga los datos de pasajeros desde un archivo JSON y los encola en una estructura de cola.

Funcionamiento:

- Solicita el nombre del archivo JSON.
- Abre el archivo y lee su contenido.
- Itera sobre cada pasajero y lo encola.

```
void consultarPasajero(Cola& colaPasajeros, ListaPasajeros& listaPasajeros) {
    string numeroPasaporte;
    cout << "Ingrese el numero de pasaporte del pasajero: ";
    cin >> numeroPasaporte;

    Cola colaAuxiliar;
    bool encontrado = false;

    while (!colaPasajeros.estaVacia()) {
        Pasajero pasajero = colaPasajeros.obtenerFrente();
        colaPasajeros.desencolar();
        colaAuxiliar.encolar(pasajero);

        if (pasajero.numero_de_pasaporte == numeroPasaporte) {
            encontrado = true;
            cout << "----------------------------" << endl;
            cout << "Pasajero encontrado:" << endl;
            pasajero.mostrar();
            cout << "----------------------------" << endl;
        }
    }

    while (!colaAuxiliar.estaVacia()) {
        colaPasajeros.encolar(colaAuxiliar.obtenerFrente());
        colaAuxiliar.desencolar();
    }

    if (listaPasajeros.contienePasajero(numeroPasaporte)) {
        encontrado = true;
        cout << "----------------------------" << endl;
        cout << "Pasajero encontrado:" << endl;
        listaPasajeros.mostrarPasajero(numeroPasaporte);
        cout << "----------------------------" << endl;
    }

    if (!encontrado) {
        cout << "No se encontro ningun pasajero con el numero de pasaporte proporcionado." << endl;
    }
}
```

Consulta los datos de un pasajero específico a partir de su número de pasaporte.

Funcionamiento:

- Solicita el número de pasaporte.
- Busca el pasajero en la cola y en la lista.
- Muestra la información del pasajero si se encuentra.

```
void cargarMovimientos(Listadoble& listaDisponibles, Listadoble& listaMantenimiento, Cola& colaPasajeros, Pila& pilaEquipajes, ListaPasajeros& listaPasajeros) {
    string nombreArchivo;
    cout << "Ingrese el nombre del archivo de movimientos: ";
    cin >> nombreArchivo;

    ifstream inputFile(nombreArchivo);
    if (!inputFile.is_open()) {
        cerr << "Error abriendo el archivo de movimientos." << endl;
        return;
    }

    string linea;
    while (getline(inputFile, linea)) {
        linea.erase(0, linea.find_first_not_of(" \t\n\r"));
        linea.erase(linea.find_last_not_of(" \t\n\r") + 1);
        if (linea == "IngresoEquipajes;") {
            cout << "Comando de ingreso de equipajes detectado." << endl;
            while (!colaPasajeros.estaVacia()) {
                Pasajero pasajero = colaPasajeros.obtenerFrente();
                colaPasajeros.desencolar();
                if (pasajero.equipaje_facturado > 0) {
                    pilaEquipajes.push(pasajero.equipaje_facturado);
                }
                listaPasajeros.insertarOrdenado(pasajero);
            }
            // Mostrar pila de equipajes
            //cout << "Pila de equipajes (de abajo hacia arriba): ";
            //pilaEquipajes.mostrar();

            // Mostrar lista de pasajeros
            //cout << "Lista de pasajeros:" << endl;
            //listaPasajeros.mostrar();
        } else if (linea.find("MantenimientoAviones,Ingreso,") == 0) {
            cout << "Comando de ingreso de aviones detectado." << endl;
            size_t inicio = 29;
            size_t fin = linea.find(":", inicio);
            string numeroRegistro = linea.substr(inicio, fin - inicio);
```

Carga y procesa los movimientos desde un archivo de texto.

Funcionamiento:

- Solicita el nombre del archivo de movimientos.
- Abre el archivo y procesa cada línea de comando.
- Maneja comandos para ingreso de equipajes y movimientos de aviones.

```
int main() {
    int opcion;
    Cola colaPasajeros;
    Listadoble listaDisponibles(nombre: "Disponibles");
    Listadoble listaMantenimiento(nombre: "Mantenimiento");
    Pila pilaEquipajes;
    ListaPasajeros listaPasajeros;

    do {
        cout << "---------- MENU ----------" << endl;
        cout << "|1. Carga de aviones     |" << endl;
        cout << "|2. Carga de pasajeros   |" << endl;
        cout << "|3. Carga de movimientos |" << endl;
        cout << "|4. Consultar pasajero   |" << endl;
        cout << "|5. Visualizar reportes  |" << endl;
        cout << "|6. Salir                |" << endl;
        cout << "--------------------------" << endl;
        cout << "Ingrese su opcion: ";
        cin >> opcion;
```

A estas opciones se accede escribiendo el numero que el usuario desea en consola

Hay varias clases, pila, cola, listadoble y listapasajeros. Tienen más o menos la misma lógica asi que a continuación se explicara la lista doble.

Constructor y Destructor

```
Listadoble::Listadoble(std::string nombre) : inicio(nullptr), nombreLista(nombre) {}

Listadoble::~Listadoble() {
    while (!estaVacia()) {
        eliminar(inicio->avion.numero_de_registro);
    }
}
```

Métodos

```
bool Listadoble::estaVacia() const {
    return inicio == nullptr;
}
```

Verifica si la lista está vacía.

```
void Listadoble::insertarAlFinal(Avion avion) {
    NodoAvion* nuevo = new NodoAvion();
    nuevo->avion = avion;
    nuevo->siguiente = nullptr;
    nuevo->anterior = nullptr;

    if (estaVacia()) {
        inicio = nuevo;
    } else {
        NodoAvion* actual = inicio;
        while (actual->siguiente != nullptr) {
            actual = actual->siguiente;
        }
        actual->siguiente = nuevo;
        nuevo->anterior = actual;
    }
}
```

Inserta un avión al final de la lista.

```cpp
void Listadoble::mostrar() const {
    if (estaVacia()) {
        std::cout << "La lista de " << nombreLista << " está vacía." << std::endl;
        return;
    }

    NodoAvion* actual = inicio;
    while (actual != nullptr) {
        std::cout << "Número de registro: " << actual->avion.numero_de_registro << ", Est
        actual = actual->siguiente;
    }
}
```

Muestra todos los aviones en la lista.

```cpp
void Listadoble::moverAvion(Listadoble& otraLista, std::string numero_de_registro) {
    NodoAvion* actual = inicio;
    while (actual != nullptr) {
        if (actual->avion.numero_de_registro == numero_de_registro) {
            Avion avion = actual->avion;
            if (nombreLista == "Disponibles") {
                avion.estado = "Mantenimiento";
            } else if (nombreLista == "Mantenimiento") {
                avion.estado = "Disponible";
            }

            eliminar(numero_de_registro);
            otraLista.insertarAlFinal(avion);
            std::cout << "Avión movido exitosamente a la lista de " << otraLista.nombreLi
            return;
        }
        actual = actual->siguiente;
    }

    std::cout << "No se encontró ningún avión con el número de registro proporcionado en
}
```

Mueve un avión de la lista actual a otra lista, actualizando su estado.

```cpp
void Listadoble::eliminar(std::string numero_de_registro) {
    if (estaVacia()) {
        std::cout << "La lista de " << nombreLista << " está vacía, no se puede eliminar
        return;
    }

    NodoAvion* actual = inicio;
    while (actual != nullptr) {
        if (actual->avion.numero_de_registro == numero_de_registro) {
            if (actual == inicio) {
                inicio = actual->siguiente;
                if (inicio != nullptr) {
                    inicio->anterior = nullptr;
                }
                delete actual;
            } else {
                actual->anterior->siguiente = actual->siguiente;
                if (actual->siguiente != nullptr) {
                    actual->siguiente->anterior = actual->anterior;
                }
                delete actual;
            }
            return;
        }
        actual = actual->siguiente;
    }

    std::cout << "No se encontró ningún avión con el número de registro proporcionado en
}
```

Elimina un avión de la lista basado en su número de registro.

```cpp
bool Listadoble::contieneAvion(const std::string& numero_de_registro) const {
    NodoAvion* actual = inicio;
    while (actual != nullptr) {
        if (actual->avion.numero_de_registro == numero_de_registro) {
            return true;
        }
        actual = actual->siguiente;
    }
    return false;
}
```

Verifica si un avión con el número de registro especificado está en la lista.

```cpp
void Listadoble::graficar(const std::string& archivo) {
    if (estaVacia()) {
        std::cout << "Lista sin elementos" << std::endl;
        return;
    }

    std::ofstream archivoSalida(archivo + ".dot");
    if (archivoSalida.is_open()) {
        archivoSalida << "digraph G {" << std::endl;
        archivoSalida << "rankdir = LR;" << std::endl;
        archivoSalida << "node [shape = record];" << std::endl;

        NodoAvion* actual = inicio;
        int i = 0;
        while (actual != nullptr) {
            archivoSalida << "node" << i << " [label = \"Número de Registro: " << actual-
            actual = actual->siguiente;
            i++;
        }

        for (int j = 0; j < i; j++) {
            archivoSalida << "node" << j << " -> node" << (j + 1) % i << ";" << std::endl
            archivoSalida << "node" << (j + 1) % i << " -> node" << j << ";" << std::endl
        }

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

        std::string comando = "dot -Tpng " + archivo + ".dot -o " + archivo + ".png";
        int resultado = system(comando.c r());
```

Genera un archivo gráfico en formato DOT para visualizar la lista y lo convierte a PNG.

En resumen, la clase Listadoble proporciona una implementación completa para manejar una lista doblemente enlazada de aviones. Incluye métodos para insertar, eliminar, mover, y mostrar aviones, así como para verificar la existencia de un avión y para generar representaciones gráficas de la lista.