Manual Técnico

1. Inclusión de Bibliotecas y Definición de Estructuras

```cpp
#include <iostream>
#include <fstream>
#include <json.hpp>
#include "Avion.h"
#include "Piloto.h"
#include "ArbolB.h"
#include "ListaCircularDoble.h"
#include "ArbolBBusqueda.h"
#include "TablaHash.h"
#include "Grafo.h"
#include "MatrizDispersa.h"
#include "ListaAviones.h"
#include "ListaPilotos.h"

using namespace std;
using json = nlohmann::json;
```

Inclusión de Bibliotecas: Se incluyen varias bibliotecas estándar de C++ y archivos de cabecera personalizados para manejar diferentes estructuras de datos y operaciones.

Uso de JSON: Se incluye la biblioteca json.hpp para manejar archivos JSON.

2. Declaración de Estructuras de Datos Globales

```cpp
ArbolBinarioBusqueda arbolPilotos;
ArbolB arbolDisponibles(3);
ListaCircularDoble listaMantenimiento;
TablaHash tablaHashPilotos(18);
Grafo grafo;
MatrizDispersa matrizDispersa;
ListaAviones listaTodosAviones;
ListaPilotos listaTodosPilotos;
```

Se declaran varias estructuras de datos globales que se usarán en el programa:

- ArbolBinarioBusqueda para almacenar pilotos.
- ArbolB para aviones disponibles.
- ListaCircularDoble para aviones en mantenimiento.
- TablaHash para pilotos.
- Grafo para rutas de vuelo.
- MatrizDispersa para relacionar aviones, pilotos y ciudades destino.
- ListaAviones y ListaPilotos para almacenar todos los aviones y pilotos cargados.

3. Funciones de Carga de Datos

3.1. Carga de Aviones

```
void cargarAviones() {
    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.capacidad = avionJson["capacidad"];
        avion.aerolinea = avionJson["aerolinea"];
        avion.ciudad_destino = avionJson["ciudad_destino"];
        avion.estado = avionJson["estado"];

        if (avion.estado == "Disponible") {
            arbolDisponibles.insertar(avion);
        } else if (avion.estado == "Mantenimiento") {
            listaMantenimiento.insertar(avion);
        }

        listaTodosAviones.insertar(avion);
    }
```

Función cargarAviones: Carga datos de aviones desde un archivo JSON.

- Abre el archivo y lo lee.
- Inserta los aviones en las estructuras correspondientes (arbolDisponibles o listaMantenimiento) según su estado.
- También agrega todos los aviones a listaTodosAviones.

3.2. Carga de Pilotos

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

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

    json pilotosJson;
    inputFile >> pilotosJson;

    for (const auto& pilotoJson : pilotosJson) {
        Piloto piloto;
        piloto.nombre = pilotoJson["nombre"];
        piloto.nacionalidad = pilotoJson["nacionalidad"];
        piloto.numero_de_id = pilotoJson["numero_de_id"];
        piloto.vuelo = pilotoJson["vuelo"];
        piloto.horas_de_vuelo = pilotoJson["horas_de_vuelo"];
        piloto.tipo_de_licencia = pilotoJson["tipo_de_licencia"];

        arbolPilotos.insertar(piloto);
        tablaHashPilotos.insertar(piloto);
        listaTodosPilotos.insertar(piloto);
    }

    cout << "Pilotos cargados correctamente." << endl;
    inputFile.close();
}
```

Función cargarPilotos: Carga datos de pilotos desde un archivo JSON.

- Abre el archivo y lo lee.
- Inserta los pilotos en las estructuras correspondientes (arbolPilotos, tablaHashPilotos, y listaTodosPilotos).

4. Comparar y Agregar Datos a la Matriz Dispersa

```
void compararYAgregarDatosAMatriz() {
    NodoAvion* nodoAvion = listaTodosAviones.cabeza;
    while (nodoAvion != nullptr) {
        NodoPiloto* nodoPiloto = listaTodosPilotos.cabeza;
        bool vueloAgregado = false;
        while (nodoPiloto != nullptr && !vueloAgregado) {
            if (nodoAvion->avion.vuelo == nodoPiloto->piloto.vuelo) {
                matrizDispersa.insertar(nodoPiloto->piloto.numero_de_id, nodoAvion->avion
                vueloAgregado = true;
            }
            nodoPiloto = nodoPiloto->siguiente;
        }
        nodoAvion = nodoAvion->siguiente;
    }
    std::cout << "Datos comparados y agregados a la matriz dispersa correctamente." << st
}
```

Función compararYAgregarDatosAMatriz: Compara los vuelos de los aviones y pilotos y los agrega a la matriz dispersa.

- Recorre listaTodosAviones y listaTodosPilotos.
- Si los vuelos coinciden, inserta el piloto y el vuelo en la matrizDispersa.

5. Carga de Rutas

```
void cargarRutas() {
    string nombreArchivo;
    cout << "Ingrese el nombre del archivo de rutas: ";
    cin >> nombreArchivo;

    grafo.cargarDesdeArchivo(nombreArchivo);
    cout << "Rutas cargadas correctamente." << endl;
}
```

Función cargarRutas: Carga rutas de vuelo desde un archivo de texto.

- Utiliza la función cargarDesdeArchivo del grafo para cargar las rutas.

6. Carga de Movimientos

```
while (getline(inputFile, linea)) {
    if (linea.find("DarDeBaja(") != string::npos) {
        size_t inicio = linea.find('(') + 1;
        size_t fin = linea.find(')');
        string id = linea.substr(inicio, fin - inicio);
        arbolPilotos.eliminar(id);
        tablaHashPilotos.eliminar(id);
        matrizDispersa.eliminarPiloto(id);
    } else if (linea.find("MantenimientoAviones,Ingreso,") == 0) {
        size_t inicio = 29;
        size_t fin = linea.find(";", inicio);
        string numeroRegistro = linea.substr(inicio, fin - inicio);

        Avion* avionAMoverPtr = arbolDisponibles.obtenerAvionPorNumeroRegistro(numero

        if (arbolDisponibles.buscar(numeroRegistro)) {
            Avion avionAMover = *avionAMoverPtr;
            arbolDisponibles.eliminar(numeroRegistro);
            listaMantenimiento.insertar(avionAMover);
        } else if (listaMantenimiento.buscar(numeroRegistro)) {
            cout << "El avion con numero de registro " << numeroRegistro << " ya esta
        } else {
            cout << "El avion con numero de registro " << numeroRegistro << " no se e
        }
    } else if (linea.find("MantenimientoAviones,Salida,") == 0) {
        size_t inicio = 28;
        size_t fin = linea.find(";", inicio);
        string numeroRegistro = linea.substr(inicio, fin - inicio);
```

Función cargarMovimientos: Procesa movimientos de aviones y pilotos desde un archivo de texto.

- Elimina pilotos.
- Mueve aviones entre listas de disponibles y mantenimiento según los comandos en el archivo.

7. Recomendación de Rutas

```
void recomendarRuta() {
    string origen, destino;
    cout << "Ingrese la ciudad origen: ";
    cin >> origen;
    cout << "Ingrese la ciudad destino: ";
    cin >> destino;

    grafo.dijkstra(origen, destino);
}
```

Función recomendarRuta: Recomienda una ruta entre dos ciudades utilizando el algoritmo de Dijkstra.

8. Visualización de Reportes

```
void mostrarAvionesDisponibles() {
    cout << "Aviones Disponibles en el Arbol B:" << endl;
    arbolDisponibles.recorrer();
    cout << endl;
}

void mostrarAvionesMantenimiento() {
    cout << "Aviones en Mantenimiento en la Lista Circular Doble:" << endl;
    listaMantenimiento.mostrar();
}

void mostrarTablaHash() {
    cout << "Tabla Hash de Pilotos:" << endl;
    tablaHashPilotos.mostrar();
}

void mostrarRutas() {
    grafo.mostrarRutas();
}

void mostrarMatrizDispersa() {
    cout << "Matriz Dispersa:" << endl;
    matrizDispersa.mostrar();
}
```

Funciones de Visualización: Muestran las diferentes estructuras de datos (aviones disponibles, aviones en mantenimiento, tabla hash de pilotos, rutas, y matriz dispersa).

9. Menú Interactivo

```cpp
int main() {
    int opcion;

    do {
        cout << "-------------- MENU --------------" << endl;
        cout << "|1. Carga de aviones              |" << endl;
        cout << "|2. Carga de pilotos              |" << endl;
        cout << "|3. Carga de rutas                |" << endl;
        cout << "|4. Carga de movimientos          |" << endl;
        cout << "|5. Consulta de horas de vuelo    |" << endl;
        cout << "|6. Recomendar ruta               |" << endl;
        cout << "|7. Visualizar reportes           |" << endl;
        cout << "|8. Salir                         |" << endl;
        cout << "-----------------------------------" << endl;
        cout << "Ingrese su opcion: ";
        cin >> opcion;

        switch(opcion) {
            case 1:
                cargarAviones();
                break;
            case 2:
                cargarPilotos();
                break;
            case 3:
                cargarRutas();
                compararYAgregarDatosAMatriz();
                break;
```

Función main: Presenta un menú interactivo al usuario.

- Permite cargar aviones, pilotos, rutas, y movimientos.
- Permite recomendar rutas y visualizar reportes.
- Usa un bucle do-while para mantener el menú hasta que el usuario elija salir.