Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Estructura de Datos
Sección A

PRACTICA 1
Manual de TECNICO

Ludwing Alexander López Ortiz
201907608

INTRODUCCIÓN

En el ámbito de la informática, las aplicaciones de escritorio han demostrado ser una solución robusta y confiable para diversas necesidades, especialmente en entornos donde la conectividad a internet no siempre está disponible. Este tipo de aplicaciones, ya sean de interfaz gráfica o de consola, permiten acceder y manipular información de manera eficiente directamente desde el sistema operativo, sin depender de una conexión en línea.

El presente proyecto propone el desarrollo de un sistema de gestión diseñado específicamente para un aeropuerto. Esta aplicación será desarrollada como parte del curso de Estructuras de Datos, con el objetivo de emplear diversas estructuras como listas enlazadas, listas circulares, pilas y colas. Estas estructuras jugarán un papel fundamental en la gestión de vuelos, pasajeros, equipajes y otras características esenciales para el funcionamiento eficiente de un aeropuerto.

La importancia de este proyecto radica en la capacidad de gestionar eficazmente los recursos y operaciones del aeropuerto, optimizando procesos clave como la programación de vuelos, el seguimiento de equipajes y la gestión de pasajeros, contribuyendo así a la eficiencia y seguridad de las operaciones aeroportuarias.

OBJETIVOS

General: Aplicar los conocimientos del curso de Estructuras de Datos en la creación de soluciones de software.

Específicos

- Hacer un uso correcto de la memoria dinámica y apuntadores en el lenguaje de programación C++.
- Aplicar los conocimientos adquiridos sobre estructuras de datos lineales.
- Utilizar la herramienta graphviz para la generación de reportes.

El código implementa varias estructuras de datos y funciones para manejar información relacionada con aviones y pasajeros, incluyendo la lectura de archivos JSON, manipulación de listas y colas, y generación de reportes visuales.

Estructuras de Datos y Clases

1. **NodoCircularD**: Define un nodo para una lista circular doblemente enlazada que almacena información sobre aviones.
    - **Variables**: Contiene datos como `vuelo`, `numero_Registro`, `modelo`, etc.
    - **next** y **prev**: Punteros al siguiente y anterior nodo en la lista.
    - **Constructor**: Inicializa los datos del nodo y los punteros.

2. **CircularDoble**: Clase que maneja la lista circular doble de aviones.
    - **Variables**: `head` es el puntero al primer nodo de la lista.
    - **Métodos**:
        - `Mostrar()`: Imprime los datos de todos los nodos en la lista.
        - `agregar(...)`: Añade un nuevo nodo a la lista.
        - `eliminar(...)`: Elimina un nodo de la lista por número de registro.
        - `Reporte()`: Genera un archivo DOT para visualizar la lista como un grafo.

3. **Pasajero**: Estructura que almacena información sobre un pasajero.
    - **Variables**: `nombre`, `nacionalidad`, `numeroPasaporte`, etc.
    - **Constructor**: Inicializa los datos del pasajero.

4. **Cola**: Clase que implementa una cola de pasajeros.
    - **Variables**: `frente` y `fondo` indican los extremos de la cola.
    - **Métodos**:
        - `agregar(...)`: Agrega un pasajero al final de la cola.
        - `eliminar()`: Elimina un pasajero del frente de la cola.
        - `mostrar()`: Muestra los pasajeros en la cola.
        - `buscarPasajero(...)`: Busca y muestra un pasajero por número de pasaporte.

5. **NodoPila** y **Pila**: Estructura y clase que implementan una pila de pasajeros.
    - **NodoPila**: Nodo de la pila que contiene un pasajero y un puntero al siguiente nodo.
    - **Pila**: Clase que maneja la pila de pasajeros con métodos para agregar, eliminar y mostrar pasajeros.

6. **NodoListaDoble** y **ListaEnlazadaDoble**: Estructura y clase que implementan una lista doblemente enlazada para pasajeros.
    - **NodoListaDoble**: Nodo de la lista doble que contiene un pasajero y punteros al siguiente y anterior nodo.
    - **ListaEnlazadaDoble**: Clase que maneja la lista doblemente enlazada con métodos para agregar, eliminar y mostrar pasajeros.

Funciones Principales

- **leerJSON_Aviones()** y **leerJSON_Pasajeros()**: Leen datos de archivos JSON de aviones y pasajeros respectivamente, y agregan estos datos a las listas o colas correspondientes (`Lista1`, `Lista2`, `Pasajeros`).
- **leerArchivo()**: Lee un archivo de texto y procesa líneas específicas relacionadas con movimientos de aviones.
- **trim(std::string &str)**: Función auxiliar para eliminar espacios en blanco de una cadena de texto.

Uso en el `main()`

- El `main()` contiene un menú interactivo que permite al usuario seleccionar diferentes opciones, como cargar aviones, cargar pasajeros, consultar pasajeros y visualizar reportes.

Resumen

El código proporciona una infraestructura básica para manejar y administrar información relacionada con aviones y pasajeros utilizando estructuras de datos como listas, colas y pilas. Además, facilita la lectura de datos desde archivos JSON y la generación de visualizaciones utilizando el formato DOT.

ANEXOS

```cpp
struct NodoCircularD
{
    //VARIABLES DEL NODO
    std::string vuelo;
    std::string numero_Registro;
    std::string modelo;
    std::string fabricante;
    int ano_fabricacion;
    int capacidad;
    int peso_maximo;
    std::string aerolinea;
    std::string estado;

    //siguiente y anterior de la lista
    NodoCircularD* next;
    NodoCircularD* prev;

    //constructor del nodo
    NodoCircularD(std::string& vuelo,std::string& numero_Registro, std::string& modelo, std::string& fabricante, int ano_fabricacion, int capacidad, int peso_maximo, std::string& aerolinea, std::string& estado)
    {
        this-> vuelo = vuelo;
        this-> numero_Registro = numero_Registro;
        this-> modelo = modelo;
        this-> fabricante = fabricante;
        this-> ano_fabricacion = ano_fabricacion;
        this-> capacidad = capacidad;
        this-> peso_maximo = peso_maximo;
        this-> aerolinea = aerolinea;
        this-> estado = estado;

        this->next = nullptr;
        this->prev = nullptr;
    }


};
```

```cpp
struct Pasajero {

    //
    std::string nombre;
    std::string nacionalidad;
    std::string numeroPasaporte;
    std::string vuelo;
    int asiento;
    std::string destino;
    std::string origen;
    int equipajeFacturado;

    //siguiente en la cola
    Pasajero* siguiente;

    //construcctor
    Pasajero(std::string nombre, std::string nacionalidad, std::string numeroPasaporte, std::string vuelo,int asiento, std::string destino, std::string origen, int equipajeFacturado)
    {
        this-> nombre = nombre;
        this-> nacionalidad = nacionalidad;
        this-> numeroPasaporte = numeroPasaporte;
        this-> vuelo = vuelo;
        this-> asiento = asiento;
        this-> destino = destino;
        this-> origen = origen;
        this-> equipajeFacturado = equipajeFacturado;

        this-> siguiente = nullptr;
    }
};
```

```cpp
json leerJSON_Aviones() {

    std::string documento; //variable con nombre del documento

    std::cout << "INGRESE LA RUTA DEL ARCHIVO: " << std::endl;
    std::cin >> documento; //INGRESAR EL NUMERO SELECCIONADO DE LA OPCION

    std::cout << "ruta: " <<documento<< std::endl;

    std::ifstream inputFile(documento);
    json jsonData;

    if (inputFile.is_open()) {
        inputFile >> jsonData;
        inputFile.close();
    } else {
        std::cerr << "No se pudo abrir el archivo: " << std::endl;
        throw std::runtime_error("Archivo no encontrado");
    }

     // Mostrar el contenido del JSON
    //std::cout << jsonData.dump(4) << std::endl;

    // Acceder a los elementos del JSON
    for (const auto& aviones : jsonData) {

        //OBTENER LOS ELEMENTOS DEL JSON AVIONES
        std::string vuelo = aviones["vuelo"];
        std::string numero_Registro = aviones["numero_de_registro"];
        std::string modelo = aviones["modelo"];
        std::string fabricante = aviones["fabricante"];
        int ano_fabricacion = aviones["ano_fabricacion"];
        int capacidad = aviones["capacidad"];
        int peso_maximo = aviones["peso_max_despegue"];
        std::string aerolinea = aviones["aerolinea"];
        std::string estado = aviones["estado"];

        /*

        std::cout<< "------------------------------------------------------------"<< std::endl;
        std::cout<< vuelo<< std::endl;
        std::cout<< numero_Registro<< std::endl;
        std::cout<< modelo<< std::endl;
        std::cout<< fabricante<< std::endl;
        std::cout<< ano_fabricacion<< std::endl;
        std::cout<< capacidad<< std::endl;
        std::cout<< peso_maximo<< std::endl;
        std::cout<< aerolinea<< std::endl;
        std::cout<< estado<< std::endl;
        std::cout<< "------------------------------------------------------------"<< std::endl;

        */

        //AGREGAR LOS ELEMENTOS DEL JSON A LA LISTA DOBLE CIRCULAR
        //Aviones.agregar(vuelo, numero_Registro, modelo, fabricante, ano_fabricacion, capacidad, peso_maximo, aerolinea, estado);


        if (estado == "Disponible")
        {
            Lista1.agragar(vuelo, numero_Registro, modelo, fabricante, ano_fabricacion, capacidad, peso_maximo, aerolinea, estado);
        }else
        {
            Lista2.agragar(vuelo, numero_Registro, modelo, fabricante, ano_fabricacion, capacidad, peso_maximo, aerolinea, estado);
        }



    }

    Lista1.Mostrar();

    Lista2.Mostrar();

     return jsonData; // Retornar los datos JSON leídos

}
```

```cpp
int main(int argc, char const *argv[])
{

    bool exit = true; // VARIABLE DE SALIDA DEL CICLO
    int seleccion; //variable seleccion de menu
    std::string pasaporte; //variable con numero de pasaporte



    while (exit)
    {
        std::cout << "||---------- Menu ----------||" << std::endl;
        std::cout << "|| 1. CARGAR AVIONES.       ||" << std::endl; //CARGA COMPLETADA
        std::cout << "|| 2. CARGAR PASAJEROS.     ||" << std::endl; //carga completa
        std::cout << "|| 3. CARGA DE MOVIMIENTOS. ||" << std::endl; //completo
        std::cout << "|| 4. CONSULTA PASAJEROS.   ||" << std::endl; //consulta completa
        std::cout << "|| 5. VISUALIZAR REPORTES.  ||" << std::endl;
        std::cout << "|| 6. SALIR.                ||" << std::endl; //funciona

        std::cin >> seleccion; //INGRESAR EL NUMERO SELECCIONADO DE LA OPCION

        if (seleccion == 6)
        {
            exit = false;
        }

        switch (seleccion)
        {
        case 1:
            try {
                    json avionesData = leerJSON_Aviones();
            } catch (const std::runtime_error& e) {
                    std::cerr << "Error: " << e.what() << std::endl;
                return 1;
            }

            break;
        case 2:
            try {
                    json pasajerosData = leerJSON_Pasajeros();
            } catch (const std::runtime_error& e) {
                    std::cerr << "Error: " << e.what() << std::endl;
                return 1;
            }
            break;
        case 3:
            /* code */
            break;
        case 4:

            std::cout << "INGRESE EL NUMERO DE PASAPORTE: " << std::endl;
            std::cin >> pasaporte; //INGRESAR EL NUMERO SELECCIONADO DE LA OPCION

            std::cout << "BUSCANDO EL PASAPORTE: " <<pasaporte<< std::endl;
            Pasajeros.buscarPasajero(pasaporte);

            break;
        case 5:
            /* code */
            break;

        default:

            std::cout << " NO INGRESO UNA OPCION VALIDA! " << std::endl;

            break;
        }
    }



    return 0;
}

```

```cpp
// Función para leer un archivo de texto y mostrar su contenido
void leerArchivo() {

    std::string documento; //variable con nombre del documento

    std::cout << "INGRESE LA RUTA DEL ARCHIVO: " << std::endl;
    std::cin >> documento; //INGRESAR EL NUMERO SELECCIONADO DE LA OPCION

    std::cout << "ruta: " <<documento<< std::endl;

    std::ifstream archivo(documento);

    if (!archivo.is_open()) {
        std::cerr << "Error al abrir el archivo: " << documento << std::endl;
        return;
    }

    std::string linea;
    while (std::getline(archivo, linea)) {
        std::cout << linea << std::endl; //lee la linea

        std::string lineaProcesada = linea;

        // Eliminar espacios
        trim(lineaProcesada);

        // Verificar y procesar la línea
        if (lineaProcesada == "IngresoEquipajes;") {

            std::cout << "Línea de tipo: IngresoEquipajes" << std::endl;

        } else if (lineaProcesada.substr(0, 19) == "MantenimientoAviones") {
        // Separar los campos
        std::stringstream ss(lineaProcesada);
        std::string segmento;
        std::vector<std::string> segmentos;

        while (std::getline(ss, segmento, ',')) {
            segmentos.push_back(segmento);
        }

        // Verificar que la línea tenga los 3 segmentos esperados y termine con ';'
        if (segmentos.size() == 3 && segmentos[2].back() == ';') {
            segmentos[2].pop_back(); // Eliminar el ';' final

            // Identificar si es Ingreso o Salida
            std::string tipo = segmentos[1];
            std::string identificador = segmentos[2];

            if (tipo == "Ingreso" || tipo == "Salida") {
                std::cout << "Línea de tipo: MantenimientoAviones" << std::endl;
                std::cout << "Tipo: " << tipo << std::endl;
                std::cout << "Identificador: " << identificador << std::endl;
            } else {
                std::cout << "Línea desconocida o con formato incorrecto" << std::endl;
            }
        } else {
            std::cout << "Línea desconocida o con formato incorrecto" << std::endl;
        }
    } else {
        std::cout << "Línea desconocida o con formato incorrecto" << std::endl;
    }


    }

    archivo.close();
}

```