UNIVERSIDAD NACIONAL DEL ALTIPLANO – EP INGENIERÍA DE SISTEMAS

CURSO: Algoritmos y Estructuras de Datos
SEMANA: 4
NIVEL: Avanzado
TEMA: Implementación avanzada de estructuras dinámicas con listas enlazadas personalizadas
DOCENTE: Mg. Aldo Hernán Zanabria Gálvez

TÍTULO DE LA PRÁCTICA

"Sistema de Gestión de Registros Académicos con Estructuras Dinámicas y Programación Orientada a Objetos"

I. OBJETIVO GENERAL

Implementar un sistema de gestión de registros basado en listas doblemente enlazadas o circulares, haciendo uso de técnicas de programación orientada a objetos, con almacenamiento y recuperación desde archivos.

II. OBJETIVOS ESPECÍFICOS

- Aplicar conceptos de encapsulamiento, abstracción y modularidad en la implementación de clases.
- Desarrollar funciones CRUD (crear, leer, actualizar, eliminar) sobre una estructura dinámica.
- Integrar lectura y escritura de archivos en formato .txt o .csv.
- Analizar la complejidad computacional de cada operación implementada.
- Reflexionar sobre la aplicabilidad real de las estructuras implementadas.

III. DESCRIPCIÓN DEL PROBLEMA

Se desea construir un sistema que permita gestionar registros de estudiantes, cada uno con los siguientes atributos:

- ID (entero autoincremental)
- Nombre completo
- Correo electrónico
- Escuela profesional
- Año de ingreso

Los registros deben ser almacenados dinámicamente mediante una estructura de lista enlazada, y deben implementarse como una clase de tipo Lista, Pila o Cola.

La información debe almacenarse además en archivos planos (.txt o .csv) para ser recuperada al reiniciar el programa.

Se sugiere implementar esta práctica en **C++**, por ser el lenguaje base del curso, aunque también puede explorarse la implementación paralela en **Python**.

IV. FUNCIONALIDADES REQUERIDAS

1. **Agregar nuevo estudiante** con ID automático.
2. **Mostrar todos los estudiantes** registrados desde el nodo inicial.
3. **Buscar estudiante** por ID o nombre.
4. **Modificar datos** de un estudiante encontrado.
5. **Eliminar un estudiante** dado su ID.
6. **Guardar los datos** en archivo .txt o .csv.
7. **Cargar los datos** desde archivo al iniciar el programa.

```
// Clase ListaDoble con operaciones básicas
class ListaDoble {
private:
    Nodo* cabeza;
    int contadorId;

public:
    ListaDoble() {
        cabeza = nullptr;
        contadorId = 1;
    }

    void agregar(string nombre, string correo, string escuela, int
anio) {
        Nodo* nuevo = new Nodo(contadorId++, nombre, correo, escuela,
anio);
        if (cabeza == nullptr) {
            cabeza = nuevo;
        } else {
            Nodo* actual = cabeza;
            while (actual->siguiente != nullptr) {
                actual = actual->siguiente;
            }
            actual->siguiente = nuevo;
            nuevo->anterior = actual;
        }
    }

    void mostrar() {
        Nodo* actual = cabeza;
        while (actual != nullptr) {
            cout << actual->id << " - " << actual->nombre << " - " <<
actual->correo << endl;
            actual = actual->siguiente;
        }
    }

    // Agregar funciones: buscar, modificar, eliminar, guardar y
cargar
};
```

VI. ACTIVIDADES COMPLEMENTARIAS

1. Elaborar un análisis de complejidad (Big-O) para cada operación principal.
2. Comparar la estructura utilizada (Listas, pilas o colas) con arreglos dinámicos en una tabla de ventajas/desventajas.
3. Proponer dos casos reales donde estas estructuras puedan ser aplicadas (por ejemplo: sistemas de turnos, gestión de memoria en sistemas operativos, listas de reproducción, etc.).

VII. PRODUCTOS A ENTREGAR

- Archivos fuente .cpp con la implementación completa.
- Archivo de texto .txt o .csv con registros simulados.
- Capturas de pantalla con pruebas de inserción, eliminación, modificación y búsqueda.
- Informe técnico en Word o PDF con:
    - Explicación del código.
    - Tabla de comparación entre estructuras.
    - Análisis de complejidad.
    - Aplicaciones reales.
    - Bibliografía en formato APA 7.

VIII. SUGERENCIA DE EXTENSIÓN

- Realizar la misma implementación en **Python**, utilizando clases y listas dinámicas.
- Aplicar principios de herencia si se desea extender los tipos de registros (por ejemplo, docente y estudiante).
- Crear un menú interactivo en consola que permita operar el sistema dinámicamente.

IX. EVALUACIÓN

Criterio	Puntaje
Aplicación del paradigma POO	4 puntos
Funcionalidad completa del sistema	6 puntos
Lectura/escritura de archivos	2 puntos
Análisis de complejidad y documentación técnica	4 puntos
Reflexión sobre aplicaciones y extensión	2 puntos
Calidad del código y presentación final	2 puntos

Puntaje total: 20 puntos

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>

using namespace std;

// Clase Nodo para Lista Doblemente Enlazada
class Nodo {
public:
    int id;
    string nombre, correo, carrera;
    int anio;
    Nodo* siguiente;
    Nodo* anterior;

    Nodo(int _id, string _nombre, string _correo, string _carrera, int
_anio)
        : id(_id), nombre(_nombre), correo(_correo),
carrera(_carrera), anio(_anio),
          siguiente(nullptr), anterior(nullptr) {}
};

// Clase ListaDoble
class ListaDoble {
private:
    Nodo* cabeza;
    int idActual;

public:
    ListaDoble() {
        cabeza = nullptr;
        idActual = 1;
        cargarDesdeArchivo();
    }

    void agregar(string nombre, string correo, string carrera, int
anio) {
        Nodo* nuevo = new Nodo(idActual++, nombre, correo, carrera,
anio);
        if (!cabeza) {
            cabeza = nuevo;
        } else {
            Nodo* actual = cabeza;
            while (actual->siguiente) actual = actual->siguiente;
            actual->siguiente = nuevo;
            nuevo->anterior = actual;
        }
        guardarEnArchivo();
    }

    void mostrar() {
        Nodo* actual = cabeza;
        cout << "\nID\tNombre\t\tCorreo\t\t\tCarrera\t\tAño\n";
        cout << "-----------------------------------------------------
---------\n";
        while (actual) {
```

```
            cout << actual->id << "\t" << actual->nombre << "\t\t" <<
actual->correo
                 << "\t" << actual->carrera << "\t" << actual->anio <<
endl;
            actual = actual->siguiente;
        }
        cout << endl;
    }

    void buscarPorNombre(string nombreBuscar) {
        Nodo* actual = cabeza;
        bool encontrado = false;
        while (actual) {
            if (actual->nombre == nombreBuscar) {
                cout << "\nRegistro encontrado:\n";
                cout << "ID: " << actual->id << ", Nombre: " <<
actual->nombre
                     << ", Correo: " << actual->correo << ", Carrera:
" << actual->carrera
                     << ", Año: " << actual->anio << endl;
                encontrado = true;
                break;
            }
            actual = actual->siguiente;
        }
        if (!encontrado)
            cout << "No se encontró el nombre: " << nombreBuscar <<
endl;
    }

    void eliminar(int idEliminar) {
        Nodo* actual = cabeza;
        while (actual) {
            if (actual->id == idEliminar) {
                if (actual->anterior) actual->anterior->siguiente =
actual->siguiente;
                else cabeza = actual->siguiente;

                if (actual->siguiente) actual->siguiente->anterior =
actual->anterior;

                delete actual;
                guardarEnArchivo();
                cout << "Registro eliminado correctamente.\n";
                return;
            }
            actual = actual->siguiente;
        }
        cout << "ID no encontrado.\n";
    }

    void guardarEnArchivo() {
        ofstream archivo("registros.txt");
        Nodo* actual = cabeza;
        while (actual) {
            archivo << actual->id << ";" << actual->nombre << ";" <<
actual->correo
```

```
                    << ";" << actual->carrera << ";" << actual->anio
<< endl;
            actual = actual->siguiente;
        }
        archivo.close();
    }

    void cargarDesdeArchivo() {
        ifstream archivo("registros.txt");
        string linea;
        while (getline(archivo, linea)) {
            stringstream ss(linea);
            string campo;
            int id, anio;
            string nombre, correo, carrera;

            getline(ss, campo, ';'); id = stoi(campo);
            getline(ss, nombre, ';');
            getline(ss, correo, ';');
            getline(ss, carrera, ';');
            getline(ss, campo, ';'); anio = stoi(campo);

            Nodo* nuevo = new Nodo(id, nombre, correo, carrera, anio);
            if (!cabeza) {
                cabeza = nuevo;
            } else {
                Nodo* actual = cabeza;
                while (actual->siguiente) actual = actual->siguiente;
                actual->siguiente = nuevo;
                nuevo->anterior = actual;
            }
            if (id >= idActual) idActual = id + 1;
        }
        archivo.close();
    }

    void recargarDesdeArchivo() {
        Nodo* actual = cabeza;
        while (actual) {
            Nodo* temp = actual;
            actual = actual->siguiente;
            delete temp;
        }
        cabeza = nullptr;
        idActual = 1;
        cargarDesdeArchivo();
        cout << "Registros recargados desde archivo correctamente.\n";
    }
};

// Función principal con menú interactivo
int main() {
    ListaDoble lista;
    int opcion;

    while (true) {
        cout << "\n--- SISTEMA DE REGISTRO ACADÉMICO ---\n";
```

```
        cout << "1. Agregar nuevo estudiante\n";
        cout << "2. Mostrar registros\n";
        cout << "3. Buscar por nombre\n";
        cout << "4. Eliminar por ID\n";
        cout << "5. Salir\n";
        cout << "6. Cargar registros desde archivo (manual)\n";
        cout << "Seleccione una opción: ";
        cin >> opcion;
        cin.ignore();

        if (opcion == 1) {
            string nombre, correo, carrera;
            int anio;
            cout << "Nombre completo: ";
            getline(cin, nombre);
            cout << "Correo: ";
            getline(cin, correo);
            cout << "Carrera profesional: ";
            getline(cin, carrera);
            cout << "Año de ingreso: ";
            cin >> anio; cin.ignore();
            lista.agregar(nombre, correo, carrera, anio);
        }
        else if (opcion == 2) {
            lista.mostrar();
        }
        else if (opcion == 3) {
            string nombreBuscar;
            cout << "Nombre a buscar: ";
            getline(cin, nombreBuscar);
            lista.buscarPorNombre(nombreBuscar);
        }
        else if (opcion == 4) {
            int id;
            cout << "ID a eliminar: ";
            cin >> id; cin.ignore();
            lista.eliminar(id);
        }
        else if (opcion == 5) {
            cout << "Gracias por usar el sistema.\n";
            break;
        }
        else if (opcion == 6) {
            lista.recargarDesdeArchivo();
        }
        else {
            cout << "Opción no válida. Intente nuevamente.\n";
        }
    }

    return 0;
}
```

```
#include <windows.h>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

// Estructura para registro
struct Estudiante {
    string nombre;
    string correo;
    string carrera;
    int anio;
};

vector<Estudiante> lista;

// Controles
HWND hNombre, hCorreo, hCarrera, hAnio, hLista;

// Función para agregar registro
void AgregarEstudiante(HWND hwnd) {
    char nombre[100], correo[100], carrera[100], anioStr[10];
    GetWindowText(hNombre, nombre, 100);
    GetWindowText(hCorreo, correo, 100);
    GetWindowText(hCarrera, carrera, 100);
    GetWindowText(hAnio, anioStr, 10);
    int anio = atoi(anioStr);

    Estudiante e = { nombre, correo, carrera, anio };
    lista.push_back(e);

    SendMessage(hLista, LB_ADDSTRING, 0, (LPARAM)(string(nombre) + " -
" + correo).c_str());

    SetWindowText(hNombre, "");
    SetWindowText(hCorreo, "");
    SetWindowText(hCarrera, "");
    SetWindowText(hAnio, "");
}

// Guardar en archivo
void GuardarArchivo() {
    ofstream file("registros.txt");
    for (auto& e : lista) {
        file << e.nombre << ";" << e.correo << ";" << e.carrera << ";"
<< e.anio << "\n";
    }
    file.close();
}

// Cargar desde archivo
void CargarArchivo(HWND hwnd) {
    lista.clear();
    SendMessage(hLista, LB_RESETCONTENT, 0, 0);
```

```
    ifstream file("registros.txt");
    string linea;
    while (getline(file, linea)) {
        stringstream ss(linea);
        string nombre, correo, carrera, anioStr;
        getline(ss, nombre, ';');
        getline(ss, correo, ';');
        getline(ss, carrera, ';');
        getline(ss, anioStr, ';');
        int anio = stoi(anioStr);

        Estudiante e = { nombre, correo, carrera, anio };
        lista.push_back(e);

        SendMessage(hLista, LB_ADDSTRING, 0, (LPARAM)(nombre + " - " +
correo).c_str());
    }
    file.close();
}

// Procesamiento de mensajes
LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg, WPARAM wp,
LPARAM lp) {
    switch (msg) {
    case WM_COMMAND:
        switch (wp) {
        case 1: AgregarEstudiante(hwnd); break;
        case 2: GuardarArchivo(); break;
        case 3: CargarArchivo(hwnd); break;
        }
        break;
    case WM_DESTROY:
        PostQuitMessage(0); break;
    default:
        return DefWindowProc(hwnd, msg, wp, lp);
    }
    return 0;
}

// Crear controles
void CrearControles(HWND hwnd) {
    CreateWindow("STATIC", "Nombre:", WS_VISIBLE | WS_CHILD, 20, 20,
80, 20, hwnd, NULL, NULL, NULL);
    hNombre = CreateWindow("EDIT", "", WS_VISIBLE | WS_CHILD |
WS_BORDER, 100, 20, 200, 20, hwnd, NULL, NULL, NULL);

    CreateWindow("STATIC", "Correo:", WS_VISIBLE | WS_CHILD, 20, 50,
80, 20, hwnd, NULL, NULL, NULL);
    hCorreo = CreateWindow("EDIT", "", WS_VISIBLE | WS_CHILD |
WS_BORDER, 100, 50, 200, 20, hwnd, NULL, NULL, NULL);

    CreateWindow("STATIC", "Carrera:", WS_VISIBLE | WS_CHILD, 20, 80,
80, 20, hwnd, NULL, NULL, NULL);
    hCarrera = CreateWindow("EDIT", "", WS_VISIBLE | WS_CHILD |
WS_BORDER, 100, 80, 200, 20, hwnd, NULL, NULL, NULL);
```

```
    CreateWindow("STATIC", "Año:", WS_VISIBLE | WS_CHILD, 20, 110, 80,
20, hwnd, NULL, NULL, NULL);
    hAnio = CreateWindow("EDIT", "", WS_VISIBLE | WS_CHILD |
WS_BORDER, 100, 110, 100, 20, hwnd, NULL, NULL, NULL);

    CreateWindow("BUTTON", "Agregar", WS_VISIBLE | WS_CHILD, 320, 20,
100, 30, hwnd, (HMENU)1, NULL, NULL);
    CreateWindow("BUTTON", "Guardar", WS_VISIBLE | WS_CHILD, 320, 60,
100, 30, hwnd, (HMENU)2, NULL, NULL);
    CreateWindow("BUTTON", "Cargar", WS_VISIBLE | WS_CHILD, 320, 100,
100, 30, hwnd, (HMENU)3, NULL, NULL);

    hLista = CreateWindow("LISTBOX", NULL, WS_VISIBLE | WS_CHILD |
WS_BORDER | LBS_NOTIFY,
                          20, 150, 400, 200, hwnd, NULL, NULL, NULL);
}

// Función principal WinAPI
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrev, LPSTR args, int
nCmdShow) {
    WNDCLASS wc = { 0 };
    wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hInstance = hInst;
    wc.lpszClassName = "RegistroWin";
    wc.lpfnWndProc = WindowProcedure;

    if (!RegisterClass(&wc)) return -1;

    HWND hwnd = CreateWindow("RegistroWin", "Sistema de Registro
Académico", WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                             100, 100, 470, 420, NULL, NULL, NULL,
NULL);

    CrearControles(hwnd);

    MSG msg = { 0 };
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}
```