Sugerencias de Mejora y Código para el Sistema de Registro Académico

# 1. Separación en archivos `.h` y `.cpp`

Se recomienda separar las clases en distintos archivos para mantener el orden del código y facilitar su mantenimiento.

// Nodo.h  
#ifndef NODO\_H  
#define NODO\_H  
#include <string>  
using namespace std;  
  
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);  
};  
#endif  
  
// Nodo.cpp  
#include "Nodo.h"  
  
Nodo::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) {}

# 2. Validación de entrada de datos

Antes de guardar los datos, valida que los campos no estén vacíos y que el año sea razonable.

if (nombre.empty() || correo.empty() || carrera.empty() || anio < 1950 || anio > 2100) {  
 cout << "Error: Datos inválidos. Verifique todos los campos.  
";  
 return;  
}

# 3. Búsqueda por otros campos

Puedes crear más métodos similares a `buscarPorNombre`, como `buscarPorCorreo` o `buscarPorCarrera`.

void buscarPorCorreo(string correoBuscar) {  
 Nodo\* actual = cabeza;  
 while (actual) {  
 if (actual->correo == correoBuscar) {  
 cout << "Encontrado: " << actual->nombre << endl;  
 return;  
 }  
 actual = actual->siguiente;  
 }  
 cout << "Correo no encontrado.\n";  
}

# 4. Modificar registros

Agregar opción para modificar datos de un estudiante por ID.

void modificarPorID(int idModificar) {  
 Nodo\* actual = cabeza;  
 while (actual) {  
 if (actual->id == idModificar) {  
 cout << "Nuevo nombre: "; getline(cin, actual->nombre);  
 cout << "Nuevo correo: "; getline(cin, actual->correo);  
 cout << "Nueva carrera: "; getline(cin, actual->carrera);  
 cout << "Nuevo año: "; cin >> actual->anio; cin.ignore();  
 guardarEnArchivo();  
 cout << "Registro actualizado.\n";  
 return;  
 }  
 actual = actual->siguiente;  
 }  
 cout << "ID no encontrado.\n";  
}

# 5. Ordenamiento

Método para ordenar la lista por nombre (implementación básica con burbuja).

void ordenarPorNombre() {  
 if (!cabeza) return;  
 bool cambiado;  
 do {  
 cambiado = false;  
 Nodo\* actual = cabeza;  
 while (actual->siguiente) {  
 if (actual->nombre > actual->siguiente->nombre) {  
 swap(actual->nombre, actual->siguiente->nombre);  
 swap(actual->correo, actual->siguiente->correo);  
 swap(actual->carrera, actual->siguiente->carrera);  
 swap(actual->anio, actual->siguiente->anio);  
 cambiado = true;  
 }  
 actual = actual->siguiente;  
 }  
 } while (cambiado);  
 guardarEnArchivo();  
}

# 6. Formato del archivo

Puedes guardar los datos con encabezados y separado por comas.

ofstream archivo("registros.csv");  
archivo << "ID,Nombre,Correo,Carrera,Año\n";  
Nodo\* actual = cabeza;  
while (actual) {  
 archivo << actual->id << "," << actual->nombre << "," << actual->correo << ","  
 << actual->carrera << "," << actual->anio << endl;  
 actual = actual->siguiente;  
}

# 7. Interfaz más limpia

Usa `setw` de `<iomanip>` para alinear columnas.

#include <iomanip>  
cout << left << setw(5) << "ID" << setw(20) << "Nombre" << setw(25)  
 << "Correo" << setw(20) << "Carrera" << setw(5) << "Año" << endl;

# 8. Destructor

Agregar un destructor a la clase `ListaDoble` para liberar memoria.

~ListaDoble() {  
 Nodo\* actual = cabeza;  
 while (actual) {  
 Nodo\* temp = actual;  
 actual = actual->siguiente;  
 delete temp;  
 }  
}

# 9. Control de duplicados

Evita agregar correos repetidos al validar antes de insertar.

Nodo\* actual = cabeza;  
while (actual) {  
 if (actual->correo == correo) {  
 cout << "Error: Ya existe un estudiante con este correo.\n";  
 return;  
 }  
 actual = actual->siguiente;  
}