Práctica de Laboratorio N° 05:

Implementación de Pilas usando Listas Simplemente Enlazadas

I. OBJETIVOS DE LA PRÁCTICA:

- Comprender el funcionamiento de la estructura de datos tipo pila (LIFO).
- Implementar una pila utilizando listas simplemente enlazadas en C++.
- Desarrollar operaciones básicas de una pila: insertar (push), eliminar (pop), mostrar, contar y buscar.

II. DESARROLLO DE LA PRÁCTICA:

1. Una pila es una estructura de datos en la que el último elemento en entrar es el primero en salir (Last In, First Out - LIFO). En esta práctica implementaremos una pila utilizando listas simplemente enlazadas. Se debe crear una clase llamada Pila con una estructura de nodo que almacene un número entero y un puntero al siguiente nodo. Además, se implementarán los siguientes métodos:
    - agregarPila(dato) – Inserta un nodo al inicio (cima o tope) de la pila.
    - sacarPila() – Elimina el nodo de la cima o tope de la pila.
    - mostrarPila() – Muestra los elementos desde la cima hasta el final.
    - contarElementosPila() – Devuelve el número de elementos en la pila.
    - buscarElementoPila(dato) – Retorna verdadero si el elemento existe en la pila.

Pila.cpp

```cpp
#include <iostream>
using namespace std;

struct Nodo {
    int info;
    Nodo* liga;
};

class Pila {
private:
    Nodo* tope;  // Apunta al nodo superior de la pila

public:
    Pila() {
        tope = NULL;
    }

    // Método para agregar un elemento a la pila
    void agregarPila(int dato) {
        Nodo* q = new Nodo;
        q->info = dato;
        q->liga = tope;
        tope = q;
    }

    // Método para eliminar (sacar) el elemento superior de la pila
    void sacarPila() {
        if (tope != NULL) {
            Nodo* q = tope;
            tope = q->liga;
            delete q;
        }else{
            cout << "Pila vacía, no se puede sacar elemento." << endl;
            return;
        }
    }

```

```
38      // Método para mostrar todos los elementos de la pila
39      void mostrarPila() {
40          if (tope != NULL) {
41              Nodo* q = tope;
42              cout << "Los elementos de la pila son:" << endl;
43              while (q != NULL) {
44                  cout << q->info << " ";
45                  q = q->liga;
46              }
47              cout << endl;
48          } else{
49              cout << "Pila vacía." << endl;
50              return;
51          }
52
53      }
54
55      // Método para contar los elementos de la pila
56      int contarElementosPila() {
65
66      // Método para buscar un elemento en la pila
67      bool buscarElementoPila(int dato) {
80
81      bool compararCon(Pila& pila2) {
100
101     // Destructor para liberar memoria
102     ~Pila() {
103         while (tope != NULL) {
104             sacarPila();
105         }
106     }
107 };
108 // Menú interactivo
109 void menu() {
110     Pila pila;
111     Pila pila2;
112     int opcion, dato;
113
114     do {
115         cout << "\n--- MENÚ PILA ---" << endl;
116         cout << "1. Apilar en Pila 1" << endl;
117         cout << "2. Desapilar en Pila 1" << endl;
118         cout << "3. Mostrar Pila 1" << endl;
119         cout << "4. Contar elementos de Pila 1" << endl;
120         cout << "5. Buscar un elemento en Pila 1" << endl;
121         cout << "6. Apilar en Pila 2" << endl;
122         cout << "7. Mostrar Pila 2" << endl;
123         cout << "8. Comparar Pila 1 con Pila 2" << endl;
124         cout << "0. Salir" << endl;
125         cout << "Opción: ";
126         cin >> opcion;
127
128         switch (opcion) {
129             case 1:
130                 cout << "Ingrese dato a apilar en Pila 1: ";
131                 cin >> dato;
132                 pila.agregarPila(dato);
133                 break;
134             case 2:
135                 pila.sacarPila();
136                 break;
137             case 3:
138                 pila.mostrarPila();
139                 break;
140             case 4:
141                 cout << "Cantidad de elementos en Pila 1: " << pila.contarElementosPila() << endl;
142                 break;
```

```
            case 5:
                cout << "Ingrese el elemento a buscar en Pila 1: ";
                cin >> dato;
                pila.buscarElementoPila(dato);
                cout << "¿Está el "<<dato<<"?: " << (pila.buscarElementoPila(dato) ? "Sí" : "No")<< endl;
                break;
            case 6:
                cout << "Ingrese dato a apilar en Pila 2: ";
                cin >> dato;
                pila2.agregarPila(dato);
                break;
            case 7:
                pila2.mostrarPila();
                break;
            case 8:
                if (pila.compararCon(pila2)) {
                    cout << "Las pilas son iguales." << endl;
                } else {
                    cout << "Las pilas son diferentes." << endl;
                }
                break;
            case 0:
                cout << "Programa finalizado." << endl;
                break;
            default:
                cout << "Opción inválida." << endl;
        }
    } while (opcion != 0);
}

int main() {

    setlocale(LC_ALL, "");

    menu();

    /*Pila pila;
    Pila pila2;
    pila.agregarPila(10);
    pila.agregarPila(20);
    pila.agregarPila(30);

    pila.mostrarPila();*/ // 30 20 10

    /*cout << "Total: " << pila.contarElementosPila() << endl;

    cout << "¿Está el 20?: " << (pila.buscarElementoPila(20) ? "Sí" : "No") << endl;

    pila.sacarPila();
    pila.mostrarPila(); */// 20 10

    /*pila2.agregarPila(10);
    pila2.agregarPila(20);
    pila2.agregarPila(40);
    pila2.mostrarPila();

    pila.mostrarPila();
    pila2.mostrarPila();

    if (pila.compararCon(pila2)) {
        cout << "Las pilas son iguales." << endl;
    } else {
        cout << "Las pilas son diferentes." << endl;
    }*/

    return 0;
}
```

2. Implementa el método para contar los elementos de la pila:

   int contarElementosPila ()

3. Implementa el método para buscar un elemento en la pila:

   bool buscarElementoPila(int dato)

4. Implementa un método que **compare dos pilas** y **devuelve true si ambas tienen los mismos elementos en el mismo orden** (desde la cima o tope hasta el fondo), y false en caso contrario

   Supongamos dos pilas:

   - **Pila A**: 30 → 20 → 10
   - **Pila B**: 30 → 20 → 10

   compararCon(B) devolverá true, porque ambos tienen:

   1. misma cantidad de elementos.
   2. mismos datos en el mismo orden.

5. Desarrolle un programa en C++ que permita verificar si una palabra ingresada por el usuario es un **palíndromo**, utilizando una estructura de datos tipo **pila** implementada con listas simplemente enlazadas.

   bool esPalindromo(const string& palabra)

   Ejemplos:

   - Entrada: "reconocer" → Salida: "Es palíndromo"
   - Entrada: "computadora" → Salida: "No es palíndromo"

     Otras palabras palíndromas: ana, rapar, oso, ala, dañad, seres, anilina

   Tip para resolver el ejercicio:

   Un **palíndromo** es una palabra que se lee igual de izquierda a derecha que de derecha a izquierda.

   Para ello, se deberá invertir la palabra original apilando cada uno de sus caracteres y luego desapilarlos para reconstruir la palabra invertida. Finalmente, se compararán ambas palabras para determinar si son iguales.