Práctica de Laboratorio N° 06:
Implementación de pilas usando arreglos

I. OBJETIVOS DE LA PRÁCTICA:

- Comprender los conceptos fundamentales de las estructuras de datos tipo pila, su comportamiento LIFO (Last In, First Out) y su utilidad en la resolución de problemas.
- Implementar la estructura de datos pila utilizando arreglos estáticos, aplicando los principios de la programación orientada a objetos: encapsulamiento, métodos y atributos privados, y la interacción mediante métodos públicos.
- Desarrollar operaciones básicas de una pila como apilar, desapilar, mostrar, contar, buscar y comparar, fortaleciendo la lógica algorítmica y la organización modular del código.

II. DESARROLLO DE LA PRÁCTICA:

1. Una pila es una estructura de datos lineal que sigue el principio LIFO (Last In, First Out). Se pueden implementar con arreglos o listas enlazadas. En esta práctica implementaremos una pila utilizando arreglos. Se creará una clase llamada Pila e implementarán los siguientes métodos:
    - pilaVacia()/pilaLlena() – Verifica si la pila está vacía o llena.
    - agregarPila(dato) – Inserta un elemento en la cima o tope de la pila.
    - sacarPila() – Elimina el elemento de la cima o tope de la pila.
    - mostrarPila() – Muestra los elementos desde la cima hasta la base.
    - contarElementosPila() – Devuelve el número de elementos en la pila.
    - buscarElementoPila(dato) – Retorna verdadero si el elemento existe en la pila.

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

#define MAX 100  // Capacidad máxima de la pila

class Pila {
private:
    char elementos[MAX];
    int tope;

public:
    Pila();

    bool pilaVacia();
    bool pilaLlena();
    void agregarPila(char dato);
    char sacarPila();
    char cima();
    void mostrarPila();
    int contarElementosPila();
    bool buscarElementoPila(int valor);
    bool compararCon(Pila& otra);
    bool revisarOperacionMatematica(const string& expmatematica);
    bool esPalindromo(const string& palabra);
};

// Constructor
Pila::Pila() {
    tope = -1;  // Pila vacía
}

```

```cpp
// Método para verificar si la pila está vacía
bool Pila::pilaVacia() {
    if(tope == -1){
        return true;
    }else{
        return false;
    }
}

// Método para verificar si la pila está llena
bool Pila::pilaLlena() {
    if(tope == MAX - 1){
        return true;
    }else{
        return false;
    }
}

// Método para agregar un elemento a la pila (push)
void Pila::agregarPila(char dato) {
    if (pilaLlena()) {
        cout << "Desbordamiento - Pila llena. No se puede agregar más elementos." << endl;
        return;
    }else{
        tope++;
        elementos[tope] = dato;
    }
}

// Método para eliminar el elemento superior de la pila (pop)
char Pila::sacarPila() {
    if (pilaVacia()) {
        cout << "Subdesbordamiento - Pila vacía. No se puede sacar elemento." << endl;
        return (0); // Carácter nulo
    }else{
        char dato = elementos[tope];
        tope--;
        return dato;
    }
}

// Método para ver el elemento en la cima de la pila (peek)
char Pila::cima() {
    if (!pilaVacia()) {
        return elementos[tope];
    } else {
        cout << "Pila vacía." << endl;
        return -1;
    }
}

// Método para mostrar todos los elementos de la pila
void Pila::mostrarPila() {
    if (pilaVacia()) {
        cout << "Pila vacía." << endl;
        return;
    }else{
        cout << "Elementos de la pila (de arriba hacia abajo):" << endl;
        for (int i = tope; i >= 0; i--) {
            cout << elementos[i] << " ";
        }
        cout << endl;
    }
}
```

```
98  // Método para contar elementos
99  int Pila::contarElementosPila() {
100     return tope+1;
101 }
102
103 // Método para buscar un elemento en la pila
104 bool Pila::buscarElementoPila(int dato) {
117
118 // Método para comparar con otra pila
119 bool Pila::compararCon(Pila& pila2) {
144
145 bool Pila::revisarOperacionMatematica(const string& expmatematica) {
165
166 // Menú interactivo
167 void menu() {
168     Pila pila1;
169     Pila pila2;
170     int opcion, dato;
171     do {
172         cout << "\n--- MENÚ PILA ---" << endl;
173         cout << "1. Apilar en Pila 1" << endl;
174         cout << "2. Desapilar en Pila 1" << endl;
175         cout << "3. Mostrar Pila 1" << endl;
176         cout << "4. Contar elementos de Pila 1" << endl;
177         cout << "5. Buscar un elemento en Pila 1" << endl;
178         cout << "6. Apilar en Pila 2" << endl;
179         cout << "7. Mostrar Pila 2" << endl;
180         cout << "8. Comparar Pila 1 con Pila 2" << endl;
181         cout << "0. Salir" << endl;
182         cout << "Opción: ";
183         cin >> opcion;
184
185         switch (opcion) {
186             case 1:
187                 cout << "Ingrese dato a apilar en Pila 1: ";
188                 cin >> dato;
189                 pila1.agregarPila(dato);
190                 break;
191             case 2:
192                 pila1.sacarPila();
193                 break;
194             case 3:
195                 pila1.mostrarPila();
196                 break;
197             case 4:
198                 cout << "Cantidad de elementos en Pila 1: " << pila1.contarElementosPila() << endl;
199                 break;
200             case 5:
201                 cout << "Ingrese el elemento a buscar en Pila 1: ";
202                 cin >> dato;
203                 cout << "¿Está el " << dato << "?: " << (pila1.buscarElementoPila(dato) ? "Sí" : "No") << endl;
204                 break;
205             case 6:
206                 cout << "Ingrese dato a apilar en Pila 2: ";
207                 cin >> dato;
208                 pila2.agregarPila(dato);
209                 break;
210             case 7:
211                 pila2.mostrarPila();
212                 break;
213             case 8:
214                 pila1.compararCon(pila2);
215                 break;
216             case 0:
217                 cout << "Programa finalizado." << endl;
218                 break;
219             default:
220                 cout << "Opción inválida." << endl;
221         }
222     } while (opcion != 0);
223 }
224
```

```cpp
// Función principal para probar la clase Pila
int main() {
    setlocale(LC_ALL, "");
    //menu();

    Pila pila;

    cout << "Agregando elementos a, b, c..." << endl;
    pila.agregarPila('a');
    pila.agregarPila('b');
    pila.agregarPila('c');

    pila.mostrarPila();

    cout << "Elemento en la cima: " << pila.cima() << endl;

    cout << "¿Contiene el a?: " << (pila.buscarElementoPila('a') ? "Sí" : "No") << endl;
    cout << "¿Contiene el d?: " << (pila.buscarElementoPila('d') ? "Sí" : "No") << endl;

    cout << "Sacando un elemento..." << endl;
    pila.sacarPila();

    pila.mostrarPila();

    cout << "Total de elementos: " << pila.contarElementosPila() << endl;

    // Probar comparación
    Pila pila2;
    pila2.agregarPila('a');
    pila2.agregarPila('b');
    pila2.mostrarPila();

    cout << "¿Las pilas son iguales?: " << (pila.compararCon(pila2) ? "Sí" : "No") << endl;

    string expresion;
    //cout << "Ingrese la operación matemática: ";
    //cin >> expresion;

    expresion = "((3+4)*(8+3**2))";
    //expresion = "(3+4)*(8+3**2))";
    //expresion = "((3+4)*(8+3**2)";
    //expresion = ")(3+4)*(8+3**2)(";
    //expresion = "(())())(()";

    if (pila.revisarOperacionMatematica(expresion)) {
        cout <<"La expresión matemática "<< expresion << " es CORRECTA." << endl;
    } else {
        cout <<"La expresión matemática "<< expresion << " es INCORRECTA." << endl;
    }

    return 0;
}
```

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

   bool buscarElementoPila(int dato)

3. 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.
4. 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 arreglos.

   bool esPalindromo(const string& palabra)
5. Implementa el método para convertir una expresión infija a postfija.
6. Implementa el método para convertir una expresión infija a prefija.
7. Implementa el método para validar si los paréntesis de una operación matemática están correctos:

   Ejemplo:

   expresion = ((3+4)*(8+3**2)) #correcto
   expresion = (3+4)*(8+3**2)) #incorrecto
   expresion = "((3+4)*(8+3**2) #incorrecto
   expresion = ")(3+4)*(8+3**2) #incorrecto
   expresion = "(())())(() #incorrecto