EJERCICIO 9

===Cadena.h===

```
#include <iostream>
#include <string>
#include <stdexcept>

using namespace std;

class CadenaOps
{
public:
  bool palindromo(const string &texto, int inicio, int fin);
};

bool CadenaOps::palindromo(const string &texto, int inicio, int fin)
{
  if (inicio < 0 || fin >= texto.length() || inicio > fin)
  {
    throw invalid_argument("Índices fuera de rango");
  }

  if (inicio >= fin) //CASO BASE: Si los índices se cruzan o se igualan, significa
que ya se ha comparado toda la cadena
  {
    return true;
  }

  if (tolower(texto[inicio]) != tolower(texto[fin])) //Caso recursivo: Se comparan
en las posiciones 'inicio' y 'fin'.
  {
    return false;
  }

  return palindromo(texto, inicio + 1, fin - 1);
}

/* ¿CUÁNDO ESTO SERÍA INFINITO?
    La recursión se volvería infinita si los índices no se actualizan correctamente
en cada llamada.
    Por ejemplo, si no se incrementa 'inicio' ni se decrementa 'fin', o si se omite
el caso base,
    el algoritmo entraría en un ciclo indefinido y provocaría un desbordamiento de
pila.
  ¿POR QUÉ ES UNA SOLUCIÓN NATURAL?
    La verificación de un palíndromo se basa en comparar los extremos y avanzar
hacia el centro.
    Esta lógica se adapta perfectamente a la recursión, ya que el problema se
reduce en cada paso.
  */
```

===main.cpp===

```
#include "Cadena.h"
#include <iostream>
#include <string>
#include <stdexcept>
#include <limits>

using namespace std;

int main(){
  CadenaOps cadenaOps;
  string texto;
  cout << "Ingrese una cadena de texto: ";
  getline(cin, texto);

  try {
    if(cadenaOps.palindromo(texto, 0, texto.length() - 1)) {
      cout << "La cadena es un palíndromo." << endl;
    } else {
      cout << "La cadena no es un palíndromo." << endl;
    }
  } catch (const invalid_argument& e) {
    cerr << "Error: " << e.what() << endl;
  }

  return 0;
}
```