EJERCICIO 3

===MCD.h===

```
#pragma once

#include <iostream>
#include <locale.h>

using namespace std;

template <typename T>
class MCD {
  private:
    T num1;
    T num2;
  public:
    MCD(T n1, T n2): num1(n1), num2(n2) {}

    T calculoMCD(const T& num1,const T& num2);
};

#include "MCD.tpp"
```

===MCD.tpp===

```
#include "MCD.h"
#include <iostream>
#include <locale.h>
#include <stdexcept>

template <typename T>
T MCD<T>::calculoMCD(const T &num1, const T &num2)
{
  if (num1 < 0 || num2 < 0)
  {
    throw std::invalid_argument("Los números no pueden ser negativos.");
  }
  cout << "MCD(" << num1 << ", " << num2 << ")" << endl;
  if (num2 == 0) //Caso base: Se basa en la propiedad matemática: MCD(a, 0) = a
  {
    return num1;
  }
  return calculoMCD(num2, num1 % num2); //Caso recursivo
}

/* ¿CUÁNDO ESTO SERÍA INFINITO?
La recursión se volvería infinita si si se usa 'numero + 1' en lugar de 'numero -
1'
, o si se omiten los casos base (numero == 0 o numero == 1),lo que provocaría un
desbordamiento de pila.
```

También puede suceder si se permite que 'numero' sea negativo.

¿Por qué es una solución natural?
La definición matemática del factorial es recursiva:
n! = n × (n - 1)!
Por eso, la solución recursiva es natural y refleja directamente la fórmula.
Aunque una versión iterativa puede ser más eficiente, la recursiva es útil para entender el proceso paso a paso.
*/

===main.cpp===

```
#include "MCD.h"
#include <iostream>
#include <locale.h>
#include <stdexcept>

using namespace std;

int main(){
  typedef int numeros;
  setlocale(LC_ALL, "es_ES.UTF-8");

  numeros n1, n2;
  cout << "Ingrese el primer número: ";
  cin >> n1;
  cout << "Ingrese el segundo número: ";
  cin >> n2;
  MCD<numeros> mcd(n1, n2);

  try {
    numeros resultado = mcd.calculoMCD(n1, n2);
    cout << "El MCD de " << n1 << " y " << n2 << " es: " << resultado << endl;
  } catch (const invalid_argument &e) {
    cerr << "Error: " << e.what() << endl;
  }

  return 0;
}
```