

## Práctica 1: Representación de números grandes

---

### 1. Objetivo

El objetivo es la implementación de clases y la sobrecarga de operadores en el lenguaje C++.

### 2. Entrega

Se realizará en dos sesiones de laboratorio:

- Sesión tutorada: del 4 al 6 de febrero.
- Sesión de entrega: del 10 al 13 de febrero.

### 3. Enunciado

En distintos campos de la ciencia existe la necesidad de trabajar con valores numéricos muy grandes, o muy pequeños [1]. Recientemente se han definido nuevos prefijos para nombrar a los múltiplos y submúltiplos de cualquier unidad del Sistema Internacional de Pesas y Medidas [3], [4].

En esta práctica se desea implementar tipos de datos en lenguaje C++ para manejar valores numéricos muy grandes, que excedan el rango de representación de los tipos de datos definidos en el lenguaje estándar [2]. Para ello se definen los siguientes tipos de datos utilizando la notación posicional [5]:

- `BigUnsigned`, representa los números no negativos. El rango de representación abarca cualquier número desde cero hasta el número con una cantidad de dígitos decimales que se pueda almacenar en la memoria de la máquina.
- `BigInteger`, representa los números enteros. El rango de representación abarca cualquier número entero, positivo o negativo, que se pueda almacenar en la memoria de la máquina. Se implementa a partir de un dato `BigUnsigned` y el signo.

Se definen las siguientes operaciones para el tipo de dato `BigUnsigned`:

- Constructores:

```
BigUnsigned(unsigned n = 0);  
BigUnsigned(const unsigned char* );  
BigUnsigned(const BigUnsigned&);           // Constructor de copia
```

- Asignación:

```
BigUnsigned& operator=(const BigUnsigned&);
```

- Inserción y extracción en flujo:

```
friend ostream& operator<<(ostream&, const BigUnsigned&);  
friend istream& operator>>(istream&, BigUnsigned&);
```

- Comparación:

```
bool operator==(const BigUnsigned&) const;  
friend bool operator<(const BigUnsigned&, const BigUnsigned&);
```

- Incremento/decremento:

```
BigUnsigned& operator++();          // Pre-incremento
BigUnsigned operator++(int);        // Post-incremento
BigUnsigned& operator--();          // Pre-decremento
BigUnsigned operator--(int);        // Post-decremento
```

- Operadores aritméticos:

```
friend BigUnsigned operator+(const BigUnsigned&, const BigUnsigned&);
BigUnsigned operator-(const BigUnsigned&) const;
BigUnsigned operator*(const BigUnsigned&) const;
friend BigUnsigned operator/(const BigUnsigned&, const BigUnsigned&);
BigUnsigned operator%(const BigUnsigned&) const;
```

Para el tipo de datos `BigInteger` se definen las mismas operaciones que para el tipo de datos `BigUnsigned`. Las operaciones de `BigInteger` se implementan a partir de las operaciones del objeto de tipo `BigUnsigned` que contiene.

En el tipo de datos `BigInteger` se añaden las siguientes operaciones:

- Constructor:

```
BigInteger(int n = 0);
BigInteger(const BigUnsigned&);          // Constructor de cambio de tipo
```

Se pide implementar un programa que calcule el máximo común divisor, `mcd` [6] de dos números de tipo de dato `BigInteger` utilizando el Algoritmo de Euclides [7]. Dados dos números enteros `a` y `b`, este algoritmo utiliza la operación resto de la división, `operator%`, y la siguiente definición del `mcd`:

```
mcd(a, 0) = a
mcd(a, b) = mcd(b, a%b)
```

#### 4. Notas de implementación

- Cada dígito de un número `BigUnsigned` se representará mediante un dato de tipo `unsigned char` que contiene el valor numérico del dígito decimal [0..9].
- Para visualizar un dígito decimal,  $0 \leq d < 9$ , habrá que convertirlo al correspondiente código ASCII, `'0'+d`. Y de forma similar, un dato de entrada `D` en formato ASCII, `'0' <= D < '9'`, habrá que convertirlo en el correspondiente dígito decimal `'0'+D`.
- Para almacenar los dígitos de un dato `BigUnsigned` se utilizará una estructura de datos de tamaño dinámico, de forma que pueda aumentar de tamaño si el valor a representar lo requiere. Por ejemplo, el tipo `std::vector`.
- Para implementar las operaciones aritméticas de suma y resta de números `BigUnsigned` se utilizan los algoritmos clásicos [8].
- Para implementar las operaciones de producto y cociente de números `BigUnsigned` se realizan sumas o restas sucesivas, respectivamente.

## 5. Referencias

- [1] Un pequeño paseo por los grandes números [UPV/EHU]:  
<https://culturacientifica.com/2022/11/16/un-pequeno-paseo-por-los-grandes-numeros/>
- [2] Rangos en C++ [cppreference.com]: <https://en.cppreference.com/w/cpp/language/types>
- [3] Los nuevos prefijos de peso y medida impulsados por las necesidades de almacenamiento digital [ReasonWhy.es]:  
<https://www.reasonwhy.es/actualidad/nuevos-prefijos-peso-medida-impulsados-necesidades-almacenamiento-digital>
- [4] Prefijos del Sistema Internacional [Wikipedia]:  
[https://es.wikipedia.org/wiki/Prefijos\\_del\\_Sistema\\_Internacional](https://es.wikipedia.org/wiki/Prefijos_del_Sistema_Internacional)
- [5] Notación posicional [Wikipedia]: [https://es.wikipedia.org/wiki/Notaci%C3%B3n\\_posicional](https://es.wikipedia.org/wiki/Notaci%C3%B3n_posicional)
- [6] Máximo Común Divisor [Wikipedia]:  
[https://es.wikipedia.org/wiki/M%C3%A1ximo\\_com%C3%BAn\\_divisor](https://es.wikipedia.org/wiki/M%C3%A1ximo_com%C3%BAn_divisor)
- [7] Algoritmo de Euclides [Wikipedia]: [https://es.wikipedia.org/wiki/Algoritmo\\_de\\_Euclides](https://es.wikipedia.org/wiki/Algoritmo_de_Euclides)
- [8] Algoritmos de suma y resta [Smartick.es]:  
<https://www.smartick.es/blog/matematicas/sumas-y-restas/resta-con-llevada/>