Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

**Grado en Ingeniería Informática**
**Algoritmos y Estructuras de Datos Avanzadas**

**Curso 2022-2023**

# Práctica 1: Representación de números grandes en notación posicional

## 1. Objetivo

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

## 2. Entrega

Se realizará en una sesión de entrega en el laboratorio entre el 13 y el 17 de febrero de 2023.

## 3. Enunciado

En distintos campos de la ciencia existe la necesidad de trabajar con valores numéricos muy grandes, o muy pequeños. 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 define el tipo de dato genérico `BigInt<size_t Base>` que representa números enteros utilizando la notación posicional [1]. En este sistema de numeración, el valor de un dígito depende de su posición relativa y de la `Base`, que determina el número de dígitos necesarios para escribir cualquier número. Por defecto, se utilizará el sistema decimal (base 10), aunque también es habitual utilizar los sistemas binario (base 2), octal (base 8) y hexadecimal (base 16). Así, por ejemplo:

- En base 10, el número $314_{(10)}$ = **3** * $10^2$ + **1** * $10^1$ + **4** * $10^0$ = $314_{(10)}$
- En base 2, el número $1010_{(2)}$ = **1** * $2^3$ + **0** * $2^2$ + **1** * $2^1$ + **0** * $2^0$ = $10_{(10)}$
- En base 8, el número $75_{(8)}$ = **7** * $8^1$ + **5** * $8^0$ = $61_{(10)}$
- En base 16, el número $1F_{(16)}$ = **1** * $16^1$ + **15** * $16^0$ = $31_{(10)}$

El rango de representación del tipo de dato `BigInt<size_t Base>` abarca cualquier número entero, positivo o negativo, que se pueda almacenar en la memoria de la máquina. Esto es, el límite del rango está determinado por el tamaño máximo que el sistema permite para la estructura de datos donde se almacenan los dígitos.

Se definen las siguientes operaciones para el tipo de dato `BigInt<size_t Base>`:

- Constructores:

```
BigInt(long n = 0);
BigInt(string&);
BigInt(const char* );
BigInt(const BigInt<Base>&); // Constructor de copia
```

- Asignación:

```
BigInt<Base>& operator=(const BigInt<Base>&);
```

- Inserción y extracción en flujo:

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

- Accesor:

```
int sign() const;            // Signo: 1 o -1
char operator[](int) const;  // Acceso al i-ésimo dígito
```

- Comparación:

```
friend bool operator==(const BigInt<Base>&, const BigInt<Base> &);
bool operator!=(const BigInt<Base>&) const;
friend bool operator>(const BigInt<Base>&, const BigInt<Base> &);
bool operator>=(const BigInt<Base> &) const;
friend bool operator<(const BigInt<Base>&, const BigInt<Base>&);
bool operator<=(const BigInt<Base>&) const;
```

- Incremento/decremento:

```
BigInt<Base>& operator++();      // Pre-incremento
BigInt<Base> operator++(int);    // Post-incremento
BigInt<Base>& operator--();      // Pre-decremento
BigInt<Base> operator--(int);    // Post-decremento
```

- Operadores aritméticos:

```
friend BigInt<Base> operator+(const BigInt<Base>&, const BigInt<Base>&);
BigInt<Base> operator-(const BigInt<Base> &) const;
BigInt<Base> operator-() const;
BigInt<Base> operator*(const BigInt<Base>&) const;
friend BigInt<Base> operator/(const BigInt<Base>&, const BigInt<Base>&);
BigInt<Base> operator%(const BigInt<Base>&) const;

// Potencia a^b
friend BigInt<Base> pow(const BigInt<Base>&, const BigInt<Base>&);
```

Utilizando el tipo de dato `BigInt<Base>`, se pide desarrollar un programa que implemente una calculadora de expresiones en notación polaca inversa [5]. El programa define una tabla `Board` de parejas: etiqueta alfanumérica y valor `BigInt<Base>`. Estas parejas son los operandos de la calculadora y se leen desde un fichero de entrada que tiene el siguiente formato:

- La primera fila contiene la `Base` utilizada para expresar los números `BigInt<Base>`.
- Las siguientes líneas comienzan con una etiqueta seguida de:
  - un número `BigInt<Base>`, o
  - una expresión en notación polaca inversa que utiliza las etiquetas previas como operandos y cualquiera de los operadores aritméticos definidos para el tipo `BigInt<Base>`.

Por ejemplo:

Base = 10
N1 = 442142117615672
N2 = 46651367647546
E1 = N1 N2 +
E2 = E1 N1 N2 - +

En cada paso el programa lee una línea del fichero. Si la línea contiene un operando, pareja etiqueta y número, lo almacena en la tabla `Board`. Si la línea contiene una expresión en notación polaca inversa, evalúa la expresión sustituyendo cada etiqueta por el número almacenado en la tabla `Board` y guarda en la tabla la nueva pareja, etiqueta y número resultado. Cuando termina de procesar todas las líneas del fichero de entrada, el programa genera un fichero de salida con los valores contenidos en la tabla `Board`.

Para el fichero de ejemplo anterior:

Base = 10
N1 = 442142117615672
N2 = 46651367647546
E1 = 488793485263218
E2 = 884284235231344

**4. Notas de implementación**

- Cada dígito de un número `BigInt<Base>` se representará mediante un dato `char` que contiene el valor numérico del dígito. Para visualizar un dígito hay que convertirlo al correspondiente código ASCII.
  - Si el valor del dígito `d` es menor de 10, se convierte al carácter `'0'+d`.
  - Si el valor del dígito `d` es igual o mayor de 10, y menor de 16, se convierte al carácter 'A' +d-10.
  - En otro caso no se puede visualizar.

- Para almacenar los dígitos de un número `BigInt<Base>` 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`.

- La implementación de las operaciones en el tipo `BigInt<Base>` se realizará operando en la base correspondiente.

- Los valores de la base que debe manejar el programa principal para representar los números son: binaria, octal, decimal o hexadecimal. Hay que tener en cuenta que la plantilla `BigInt<Base>` se debe expandir en tiempo de compilación.

**5. Referencias**

[1] Notación posicional [Wikipedia]: https://es.wikipedia.org/wiki/Notaci%C3%B3n_posicional

[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

[5] Notación polaca inversa [Wikipedia]:
https://es.wikipedia.org/wiki/Notaci%C3%B3n_polaca_inversa#El_algoritmo_RPN