Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2024-2025

Práctica 2: Notación posicional para números grandes

1. Objetivo

El objetivo es repasar la implementación en lenguaje C++ de las plantillas y la especialización de plantillas como mecanismo para expresar situaciones particulares.

2. Entrega

La práctica se desarrolla en dos sesiones de laboratorio:

Sesión tutorada: del 17 al 20 de febrero.
Sesión de entrega: del 24 al 27 de febrero.

3. Enunciado

En la práctica anterior [1] se han implementado tipos de datos definidos por el usuario, clases, que permiten representar y operar con valores numéricos que sobrepasan el rango de representación de los tipos básicos del lenguaje C++ [2]. Para representar estos valores numéricos se ha utilizado el sistema de numeración decimal, que es un caso particular de la notación posicional [3]. En notación posicional el valor de cada dígito depende de su posición relativa, y de una `Base` que determina la cantidad de dígitos necesarios para escribir cualquier número. En el sistema de numeración decimal se utiliza el valor de `Base=10`. También son habituales los sistemas de numeración: 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 = $31_{(10)}$

En esta práctica se definirán los tipos de datos genéricos, plantillas de clases en lenguaje C++, para representar los números grandes [1], utilizando el valor de la base especificada por el programador mediante el parámetro, `Base`, de la plantilla. Para ello se desarrollarán las siguientes plantillas:

- `BigUnsigned<unsigned char Base>`, que genera las clases para representar a los números grandes no negativos utilizando la notación posicional con el valor de la `Base` indicada en el parámetro de la plantilla. Las clases generadas por esta plantilla deben implementar, al menos, las mismas operaciones definidas en la clase `BigUnsigned` de la práctica [1].

- `BigInteger<unsigned char Base>`, que genera las clases para representar a los números grandes enteros utilizando la notación posicional con el valor de la `Base` indicada en el parámetro de la plantilla. Las clases generadas por esta plantilla deben implementar, al menos, las mismas operaciones definidas en la clase `BigInteger` de la práctica [1].

- `BigRational<unsigned char Base>`, que genera las clases para representar a los números racionales [4] utilizando una pareja de datos (`BigInteger<Base>`, `BigUnsigned<Base>`), que denotan al numerador y denominador, respectivamente. Las clases generadas por esta plantilla deben implementar, al menos, las mismas operaciones

definidas para el tipo de datos `BigUnsigned` con las siguientes modificaciones:

- Se añade el constructor:

```
BigRational(const BigInteger<Base>& numerador=0,
            const BigNatural<Base>& denominador=1);
```

- Se eliminan el constructor a partir del tipo básico `unsigned`, las operaciones de incremento/decremento y el módulo de la división.
- Para representar un número racional se utilizará la fracción irreducible, que se define como aquella en la que el numerador y el denominador no comparten factores comunes.

La representación de números enteros en el sistema binario Complemento a dos [5] permite optimizar el almacenamiento y la implementación de las operaciones. En este formato no se requiere almacenar el signo del número, pero cambia la forma de calcular la aportación de cada dígito al valor total del número respecto a la notación posicional. En complemento a dos, si `N` es un número entero almacenado con `n` dígitos, se verifica:

- El rango de valores que puede tomar está definido en $-2^{n-1} <= N <= 2^{n-1}-1$
- Si `N >= 0` entonces su bits más significativo debe ser `0`, y el valor que aporta cada dígito depende de su posición 2^i con `0 <= i < n-1`;.
- Si `N < 0` entonces su bit más significativo debe ser `1`, y el valor del número se corresponde con el valor usando la representación posicional de su complemento a dos, $C_2(N) = 2^n - N$.

La representación en complemento a dos permite calcular la resta de números enteros binarios sumando al minuendo el complemento a dos del sustraendo. Esto es, N - M = N + C_2(M).

Por otro lado, en los números en formato binario es posible almacenar los dígitos del número (bits) en un vector de `bool`, en lugar de un vector de `char` como se ha hecho en la plantilla genérica para cualquier valor de la `Base`. La librería estándar del lenguaje C++ implementa una especialización de su estructura `std::vector<bool>` [6] para optimizar el espacio de almacenamiento.

Se pide implementar un programa principal que lea desde fichero dos números racionales grandes, representados en la base indicada en el propio fichero, y los almacene en datos del tipo `BigRational<Base>`. El programa calcula cada una de las operaciones definidas para el tipo de dato y genera un fichero con los resultados:

El formato del fichero de entrada:

- La primera fila contiene la `Base` utilizada para expresar los números `BigRational<Base>`.
- Las siguientes líneas comienzan con una etiqueta seguida del símbolo `'='`, el numerador expresado como un número `BigInteger<Base>`, del símbolo `'/'`, y el denominador expresado como un número `BigUnsigned<Base>`.

Por ejemplo:

Base = 10
N1 = -442142117615672 / 46651367647546
N2 = 46651367647546 / 442142117615672

El formato del fichero de salida contiene las líneas del fichero de entrada y una línea para cada resultado. Para el ejemplo anterior:

Base = 10
N1 = -442142117615672 / 46651367647546
N2 = 46651367647546 / 442142117615672
N1 ==N2: false
N1 < N2: true
N1+N2: …

4. Notas de implementación

- Las particularidades en la implementación de la plantilla de números para la base binaria se expresa mediante una especialización en la plantilla `BigInteger<Base>`, que se define con la siguiente sintaxis:

```
template<>
class BigInteger<2> { //Código de plantilla para Base=2 en binario complemento a dos };
```

- La especialización de una plantilla forma parte de la misma plantilla donde se define el comportamiento genérico. Por tanto, el código de la especialización también se coloca en el mismo fichero cabecera de la plantilla genérica.

- Para visualizar un dígito en un sistema de numeración con valor de `10<Base<=26` se utilizarán los sucesivos códigos ASCII, desde el carácter `'A'` hasta el carácter `'Z'`. La transformación del valor numérico del dígito `d` al correspondiente carácter ASCII sigue las siguientes reglas:
  - Si el valor del dígito `d<10`, se convierte al carácter ASCII `'0'+d`.
  - Si el valor del dígito `10>=d<26`, se convierte al carácter `'A'+d-10`.
  - En otro caso, `d>=26`, no se visualizará.

- La implementación de las operaciones en las plantillas `BigUnsigned<Base>` y `BigInteger<Base>` se realizarán operando en la base indicada.

- Los valores de la base que debe manejar el programa principal para representar los números son: binaria, octal, decimal o hexadecimal.

5. Referencias

[1] Práctica 1. Números grandes [Moodle]: https://campusingenieriaytecnologia2425.ull.es/mod/resource/view.php?id=19814
[2] Rangos en C++ [cppreference.com]: https://en.cppreference.com/w/cpp/language/types
[3] Notación posicional [Wikipedia]: https://es.wikipedia.org/wiki/Notaci%C3%B3n_posicional
[4] Número racional [Wikipedia]: https://es.wikipedia.org/wiki/N%C3%BAmero_racional
[5] Complemento a dos [Wikipedia]: https://es.wikipedia.org/wiki/Complemento_a_dos
[6] `std::vector<bool>` [cpluplus.com]: https://cplusplus.com/reference/vector/vector-bool/