Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2018-2019

Práctica 2: Representación de números en notación posicional (2)

1. Objetivo

En esta práctica se trabajarán los siguientes conceptos del lenguaje C++: la especificación de clases con atributos en memoria dinámica, la especialización de plantillas como mecanismo para expresar situaciones particulares y el uso de excepciones para delegar el tratamiento de situaciones especiales.

2. Entrega

Esta práctica se realizará en dos sesiones de laboratorio en las siguientes fechas:

- Sesión tutorada: del 26 al 28 de febrero de 2019.
- Sesión de entrega: del 6 al 7 de febrero de 2019. El alumnado de los grupos PE102 y PE202, afectados por el festivo del 5 de marzo, podrán entregar la práctica en el horario de cualquiera de los otros grupos. En caso de no poder asistir a ninguno de los otros grupos, deben comunicarlo a su profesor de prácticas.

3. Enunciado

A partir de la implementación de la plantilla `Number` elaborada en la práctica 1 para representar números en notación posicional [1], se quiere generar una nueva versión de la plantilla introduciendo los siguientes cambios:

1. Se modifica el tipo del atributo privado donde se almacenan los dígitos del objeto `Number<N,B,T>`. En lugar de utilizar un `vector<T>` o un array de tipo `T`, el atributo se declara como un puntero a `T`, y el espacio para almacenar los `N` dígitos se reserva en la memoria dinámica. Este cambio implica que el constructor de una clase `Number<N,B,T>` deberá reservar el espacio en memoria dinámica y el destructor de la clase deberá liberar dicho espacio. También implica la implementación de un método constructor de copia y la sobrecarga del operador de asignación para sustituir a las versiones por defecto del compilador. Puesto que el número de dígitos en todos los objetos instanciados de una clase `Number<N,B,T>` está fijado por la constante `N`, el bloque en memoria dinámica reservado para almacenar dichos dígitos no se debe cambiar durante la existencia del objeto.

2. Se introduce un nuevo atributo privado en la plantilla para recoger el signo del valor almacenado en un objeto de una clase `Number<N,B,T>`. De esta forma, el parámetro pasado al constructor de la clase admitirá tanto valores positivos como negativos, y por supuesto el valor cero. La implementación de las operaciones deberá tener en cuenta el signo de los operandos.

   En el caso de las clases `Number<N,2,T>`, esto es, las clases que representan a los números en base binaria, se utilizará la representación de los dígitos en formato binario en complemento a 2 [2]. En esta representación el signo de un número se recoge en el bit más significativo, por tanto, no es necesario añadir otro atributo para el signo. Esta particularidad para la base binaria en la plantilla se expresa mediante una especialización parcial en la plantilla `Number`, que se utiliza la siguiente sintaxis:

```
template<size_t N, class T>
class Number<N,2,T> { //Código plantilla para B=2 };
```

Además de la diferencia en la representación del número, la utilización del formato en complemento a 2 permite implementar la operación resta como la suma del minuendo con el complemento a 2 del sustraendo.

3. En la implementación de la primera versión de la plantilla se han detectado situaciones en las que no está clara cuál debería ser la acción a realizar. Por ejemplo, cuando el número de dígitos requeridos para representar un valor en una base sobrepasa el tamaño `N` fijado en la definición de clase `Number<N,B,T>`. Otra situación en la que no está establecido un tratamiento se produce al intentar visualizar un objeto `Number<N,B,T>` cuando `B` es mayor de 16. En estas, o cualquier otra situación en las operaciones implementadas en una clase `Number<N,B,T>`, para las que no exista un tratamiento definido, se utilizará el mecanismo de manejo de excepciones del lenguaje C++.

   Todas las excepciones definidas para las clases generadas por la plantilla `Number` derivarán de la clase `NumberException`, que a su vez deriva de la clase `exception` definida en la librería estándar.

Implementar un programa principal que compruebe el correcto funcionamiento de las operaciones definidas en la plantilla `Number` para las distintas clases, en particular para las bases: binaria, decimal y hexadecimal. El programa solicitará valores al usuario para instanciar números, realizará las operaciones y mostrará por pantalla los resultados. En caso de detectar la ocurrencia de alguna excepción generada por un objeto de una clase `Number<N,B,T>` el programa principal informará al usuario y continuará con la ejecución del programa.

4. Referencias

[1] Notación posicional en Wikipedia: https://es.wikipedia.org/wiki/Notaci%C3%B3n_posicional
[2] Complemento a 2 en Wikipedia: https://es.wikipedia.org/wiki/Complemento_a_dos