Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2024-2025

Práctica 3: Calculadora para números grandes

1. Objetivo

En esta práctica se trabajarán los siguientes conceptos del lenguaje C++: la sobrecarga del operador de cambio de tipo, la herencia, el polimorfismo y el control de excepciones.

2. Entrega

Se entregará en una sesión presencial en el laboratorio entre el 10 y el 13 de marzo de 2025.

3. Enunciado

En la práctica 2 [2] se han implementado varios tipos genéricos de datos, plantillas, que permiten representar y operar números grandes en notación posicional para diferentes valores de la base. Cada tipo genérico de datos implementado (`BigUnsigned`, `BigInteger` y `BigRational`) está definido para operar con números del mismo tipo.

En esta práctica se define un nuevo tipo genérico de datos, `BigNumber<unsigned char Base>`, que generaliza el concepto de número grande y cada tipo de número grande ya implementado (`BigUnsigned`, `BigInteger` y `BigRational`) es un caso particular de número grande. De esta forma, para cada valor del parámetro de la plantilla, `Base`, se define una jerarquía de números grandes. En la clase base de la jerarquía, `BigNumber<Base>`, se declaran las operaciones comunes mediante métodos virtuales puros, y cada clase derivada debe proveer su implementación polimórfica para estos métodos virtuales, de tal forma que se puedan realizar operaciones con números grandes de tipos distintos.

En la plantilla de clases `BigNumber<Base>` se declaran, al menos, los siguientes métodos:

- Se definen las operaciones aritméticas con operandos de tipo referencia a la clase base, de forma que la decisión sobre la versión de la operación a utilizar se realiza en tiempo de ejecución según el tipo del objeto que la invoca.

```
virtual BigNumber<Base>& add(const BigNumber<Base>&) const = 0;
virtual BigNumber<Base>& subtract(const BigNumber<Base>&) const = 0;
virtual BigNumber<Base>& multiply(const BigNumber<Base>&) const = 0;
virtual BigNumber<Base>& divide(const BigNumber<Base>&) const = 0;
```

- Se definen los operadores de cambio de tipo para realizar la conversión entre los distintos tipos de números grandes.

```
virtual operator BigUnsigned<Base>() const = 0;
virtual operator BigInteger<Base>() const = 0;
virtual operator Bigrational<Base>() const = 0;
```

- Se definen métodos protegidos para realizar las operaciones de lectura y escritura de los datos de tipo `BigNumber<Base>` en objetos del tipo flujo. Los operadores de inserción y extracción en flujo se sobrecargan como funciones amigas y se implementan invocando a los correspondientes métodos virtuales.

```
virtual std::ostream& write(std::ostream&) const = 0;
virtual std::istream& read(std::istream&) = 0;
friend std::ostream& operator<<(std::ostream&, const BigNumber<Base>&);
friend std::istream& operator>>(std::istream&, BigNumber<Base>&);
```

- Se define un método estático para instanciar números grandes en memoria dinámica decidiendo el tipo particular en tiempo de ejecución. Este método recibe como parámetro una cadena de caracteres ASCII con la representación del número en la `Base` indicada, y retorna un puntero al objeto creado en memoria dinámica.

  ```
  static BigNumber<Base>* BigNumber<Base>::create(const char*);
  ```

  Para indicar el tipo particular de número grande a crear se añaden los siguientes sufijos a la cadena de caracteres:

  ```
  BigUnsigned: “1234u”
  BigInteger:  “4321i”
  BigRational: “4321/1234r”
  ```

En las implementaciones previas se han detectado algunas situaciones excepcionales, normalmente de error, para las cuales no existe una acción de tratamiento definida de antemano. Por ejemplo, ¿qué hacer cuando la entrada contiene dígitos no válidos para la `Base` de la plantilla?, ¿qué hacer ante una división por cero?, etc. En las nuevas versiones de las plantillas se utilizarán las excepciones para notificar dichas situaciones, y delegar en el código que utiliza la plantilla la decisión de cómo manejar cada situación.

Para notificar las excepciones se define una familia de clases que derivará de la clase base **`BigNumberException`**, que a su vez derivará de la clase `std::exception`. Se deben manejar, al menos, las siguientes situaciones:

- Presencia de un dígito no válido en la cadena de entrada. Esta situación puede detectarse en un constructor o en la operación de extracción de un flujo. Se notifica con la excepción `BigNumberBadDigit`.
- División por cero, que se notifica con la excepción `BigNumberDivisionByZero`.

Se pide desarrollar un programa principal que implemente una calculadora de expresiones en notación polaca inversa [3] con operandos de tipo puntero a `BigNumber<Base>`. Para almacenar los operandos, el programa define una tabla `Board` de parejas: etiqueta y operando:

```
std::pair<string, BigNumber<Base>*>
```

La entrada al programa es un fichero con el siguiente formato:

- La primera fila contiene la `Base` utilizada para representar los números grandes.
- Cada línea comienza con una etiqueta alfanumérica seguida de:
  - Un símbolo ‘=’ y una cadena de caracteres ASCII para inicializar un número grande indicando su tipo, o
  - Un símbolo ‘?’ y una expresión en notación polaca inversa que utiliza

como operandos las etiquetas leídas previamente y como operadores aritméticos los que están definidos para la clase base `BigNumber<Base>`.

Por ejemplo:

```
Base = 16
N1 = 236i
N2 = AB64u
E1 ? N2 N1 +
```

En cada paso el programa lee una línea del fichero de entrada. Si la línea contiene un operando, genera la pareja etiqueta y dato `BigNumber<Base>*` inicializado con el valor leído. Esta pareja se almacena en la tabla `Board`. Si la línea contiene una expresión en notación polaca inversa, se le pasa a una función calculadora que evalúa dicha expresión sustituyendo cada etiqueta por el objeto apuntado por el puntero `BigNumber<Base>*` almacenado en la tabla `Board`. El resultado de evaluar la expresión se guarda en la tabla junto a su etiqueta. Cuando el programa termina de procesar todas las líneas del fichero de entrada, genera un fichero de salida con los valores contenidos en la tabla `Board`.

El programa principal debe manejar las situaciones de excepción generadas. En general, la acción de tratamiento para cualquiera de las excepciones será terminar el paso actual almacenando en la tabla `Board` una pareja, etiqueta y puntero a un objeto inicializado a cero; y continuar en el siguiente paso del programa.

Para el fichero de entrada de ejemplo anterior, el fichero de salida será:

```
Base = 16
N1 = 236i
N2 = AB64u
E1 = AD9Ai
```

4. Notas de implementación

- Debido a que la clase base `BigNumber<Base>` es abstracta, no es posible instanciar objetos de dicho tipo. Por este motivo, la declaración de los métodos en esta clase requiere parámetros de tipo puntero o referencia, de manera que permita crear objetos de cualquier clase derivada y habilitar el polimorfismo dinámico.

5. Referencias

[1] Enunciado de la práctica 1 [Moodle]: enlace al aula virtual.

[2] Enunciado de la práctica 2 [Moodle]: enlace al aula virtual.

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