Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2022-2023

Práctica 3: Calculadora universal para números enteros grandes

1. Objetivo

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

2. Entrega

Se entregará en una sesión presencial en el laboratorio entre el 6 y el 10 de marzo de 2023.

3. Enunciado

A partir de la implementación de la plantilla `BigInt<size_t Base>`, elaborada en la práctica 2 [2], se quiere generar una nueva versión de la plantilla que permita realizar las operaciones cuando los operandos sean números enteros grandes en bases distintas.

Para ello se define la clase `Number`, que es la clase base de la cual derivan todas las clases generadas por la plantilla `BigInt<size_t Base>`. En esta clase base se declaran las operaciones comunes virtuales que heredan todas las clases derivadas. Cada clase derivada provee una implementación polimórfica para las operaciones virtuales.

En la clase `Number` se declaran, al menos, los siguientes métodos:

- Se definen las operaciones aritméticas con datos de tipo puntero 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 Number* add(const Number*) const = 0;
virtual Number* subtract(const Number*) const = 0;
virtual Number* multiply(const Number*) const = 0;
virtual Number* divide(const Number*) const = 0;
virtual Number* module(const Number*) const = 0;
virtual Number* pow(const Number*) const = 0;
```

- Se definen las operaciones de cambio de tipo.

```
virtual operator BigInt<2>() const = 0;
virtual operator BigInt<8>() const = 0;
virtual operator BigInt<10>() const = 0;
virtual operator BigInt<16>() const = 0;
```

- Se definen métodos protegidos para realizar la lectura y escritura de datos de tipo `Number` en objetos del tipo flujo. También se sobrecargan los operadores de inserción y extracción en flujo, que invocan 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 Number&);
friend std::istream& operator>>(std::istream&, Number&);
```

- Se define un método estático para instanciar objetos de tipo `Number` en memoria dinámica. Este método recibe como parámetros la `Base` a utilizar para instanciar un objeto `BigInt<Base>` y una cadena de caracteres ASCII con la representación del número en

dicha `Base`. Retorna un puntero al objeto creado en memoria dinámica.

```
static Number* create(size_t base, const string& s);
```

En las implementaciones previas de la plantilla `BigInt<size_t Base>` se han detectado algunas situaciones, normalmente de error, en las cuales no está definida la acción a realizar. La nueva versión de la plantilla utilizará excepciones para notificar dichas situaciones, y en consecuencia, delegar en el código cliente (que utiliza la plantilla) la decisión de cómo manejar cada situación.

Para ello se define una familia de clases que derivan de la clase base `BigIntException`, que a su vez deriva la clase `std::exception`. Se deben manejar, al menos, las siguientes situaciones:

- Presencia de un dígito no válido en la cadena de inicialización de un objeto `BigInt<Base>`. Esta situación puede detectarse en un constructor o en la operación de extracción de un flujo. Se notifica con la excepción `BigIntBadDigit`.
- División por cero, que se notifica con la excepción `BigIntDivisionByZero`.
- Intento de crear un objeto `BigInt<Base>` con un valor de `Base` no implementado. Esta situación se detecta en el método creador de instancias de objetos `Number` y se notifica con la excepción `BigIntBaseNotImplemented`.

A partir de la nueva versión de la plantilla `BigInt<size_t Base>` se pide desarrollar un programa que implemente una calculadora de expresiones en notación polaca inversa con operandos de tipo `Number`. Para almacenar los operandos el programa define una tabla `Board` de parejas: etiqueta alfanumérica y un puntero a `Number*`, que apunta al objeto creado en memoria dinámica a partir de los datos leídos desde un fichero de entrada con el siguiente formato:

- Cada línea comienza con una etiqueta alfanumérica seguida de:
    - Un símbolo `'='`, un número indicando la `Base`, un carácter separador `','` y una cadena de caracteres ASCII para inicializar un número `BigInt<Base>`, 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 `Number`.

Por ejemplo:

N1 = 2, 000100110110
N2 = 16, AB64
E1 ? N2 N1 +

En cada paso el programa lee una línea del fichero de entrada. Si la línea contiene un operando, genera una pareja etiqueta y puntero al objeto `BigInt<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 objeto apuntado desde tabla `Board`. Los números de la tabla `Board` se manejan mediante punteros a la clase base `Number`, de forma que la versión de cada operación realizada por la calculadora se determina en tiempo de ejecución en función del tipo de los operandos involucrados. 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 la plantilla

`BigInt<size_t Base>`. 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 ejemplo anterior:

N1 = 2, 000100110110
N2 = 16, AB64
E1 = 16, AC9A

4. Notas de implementación

- Dependiendo de la codificación podría ser necesario añadir a la clase `Number` la sobrecarga del operador de asignación. Esta operación requiere convertir el objeto pasado por parámetro al mismo tipo del objeto que invoca al método antes de realizar la copia de los atributos.

  ```
  virtual Number& operator=(const Number&) = 0;
  ```

- Debido a que la clase base `Number` 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 la crear objetos cualquier clase derivada y habilitar el polimorfismo dinámico.

- No es posible declarar métodos virtuales mediante una plantilla, por lo cual hay que declarar por separado cada uno de los operadores de cambio de tipo.

5. Referencias

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

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