Práctica 3: Polimorfismo y excepciones

1. Objetivo

En esta práctica se trabajan los siguientes objetivos:

1. Utilizar el polimorfismo para crear una jerarquía de clases;
2. Introducir el manejo de errores mediante el uso de excepciones.

Para realizar esta práctica se utilizará y modificará el código generado en las dos primeras prácticas: implementación de estructuras de datos e implementación de una calculadora en notación postfija.

2. Entrega

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

Sesión tutorada: 15 al 18 de marzo de 2016.
Sesión de entrega: 29 de marzo al 1 de abril de 2016.

3. Enunciado

El primer objetivo de la práctica consiste en crear una jerarquía de tipos de números. Para ello se creará una clase abstracta `Numero` que define todas las operaciones comunes de los tipos de datos numéricos. A partir de esta clase abstracta se derivan las clases que implementan las operaciones para cada uno se los tipos de números. Los tipos de números desarrollados en la práctica 2: `Racional` y `Complejo`, se amplían con la implementación de los tipos de números: `Entero` y `Real`, que encapsulan un dato de tipo básico, `int` y `double`, respectivamente.

La clase Numero define las operaciones, como mínimo:

```
class Numero {
  public:
    // Devuelve una copia del Numero actual en el tipo Entero
    virtual const Entero toEntero() const = 0;
    // Devuelve una copia del Numero actual en el tipo Racional
    virtual const Racional toRacional() const = 0;
    // Devuelve una copia del Numero actual en el tipo Real
    virtual const Real toReal() const = 0;
    // Devuelve una copia del Numero actual en el tipo Complejo
    virtual const Complejo toComplejo() const = 0;
    // Escribe un Numero al flujo sout
    virtual ostream& toStream(ostream& sout) const = 0;
    // Lee un Numero desde flujo sin
    virtual istream& fromStream(istream& sin) = 0;
};
```

```
class Entero: public Numero {
  public:
      ...
    // Devuelve una copia del Numero actual en el tipo Entero
    const Entero toEntero() const;
    // Devuelve una copia del Numero actual en el tipo Racional
    const Racional toRacional() const;
    // Devuelve una copia del Numero actual en el tipo Real
    const Real toReal() const;
    // Devuelve una copia del Numero actual en el tipo Complejo
    const Complejo toComplejo() const;
    // Escribe un Numero al flujo
    ostream& toStream(ostream& sout) const;
    // Lee un Numero desde flujo
    istream& fromStream(istream& sin);
      ...
};
```

El segundo objetivo de la práctica consiste en manejar, mediante el mecanismo de las excepciones, las condiciones de error que se detectan en las implementaciones de las plantillas (vector, Lista, Pila y cola), y en los tipos de números (entero, racional, real y complejo).

Los errores más habituales en la implementación de las estructuras de datos son los fallos en la reserva de memoria dinámica (uso del operador `new`), y los accesos a posiciones fuera de límite (posiciones fuera del vector, o extracciones en una pila vacía). En el caso de los tipos de números hay que controlar la división por cero, y los cambios de tipo que implican pérdida de precisión.

Para comprobar el funcionamiento del código escribir un programa que utilice las plantillas con datos de cualquier tipo de número a través de punteros a la clase base (`Numero*`).

Durante las sesiones de laboratorio se podrán proponer modificaciones y mejoras en el enunciado de la práctica.