Práctica 2: Calculadora de expresiones en notación postfija

1. Objetivo

El objetivo de esta práctica es trabajar la definición de clases, la implementación de operaciones mediante métodos, uso de funciones amigas y la sobrecarga de operadores. También se introduce las plantillas para definir clases genéricas.

2. Entrega

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

Sesión tutorada: 1 al 4 de marzo de 2016.
Sesión de entrega: 8 al 11(*) de marzo de 2016.

(*) El grupo del viernes (L06), afectados por el festivo del 11 de marzo que no puedan asistir a la sesión de entrega en otro grupo, deben contactar con el profesorado de prácticas.

3. Enunciado

Implementar en lenguaje C++ una calculadora que reciba como entrada una expresión aritmética en notación postfija, la evalúe y retorne su resultado.

En la notación postfija, también denominada notación polaca inversa [1], los operandos de la expresión se introducen primero y a continuación se introduce el operador. Esta notación tiene la ventaja frente a la notación infija, la notación tradicional para las expresiones aritméticas, de no requerir el uso de paréntesis para indicar el orden de las operaciones. La siguiente tabla muestra algunas expresiones aritméticas con datos enteros expresadas en las notaciones infija y postfija.

Infija	Postfija
3+2	3 2 +
3+2*5	3 2 5 * +
3*(2+5)	3 2 5 + *
(3+2)*(5-7)	3 2 + 5 7 - *

La evaluación de una expresión aritmética en notación postfija se realiza utilizando una pila para almacenar los operandos. Cuando se lee un operando de la expresión de entrada se guarda en la pila; mientras que si lo que se lee de la expresión de entrada es un operador se recuperan los operandos necesarios de la pila, se evalúa la operación y se guarda el resultado en la pila. El terminar de procesar la expresión de entrada el resultado de la misma queda en el top de la pila.

Expresión entrada	Dato leido	Dato evaluado	Pila
3 2 + 5 7 - *			[vacía]
2 + 5 7 - *	3		[3]
+ 5 7 - *	2		[3][2]
5 7 - *	+	3+2=5	[5]
7 - *	5		[5][5]
- *	7		[5][5][7]
*	-	5-7=-2	[5][-2]
	*	5*(-2)=-10	[-10]

Como primera aproximación, implementar una función calculadora que lea la expresión de entrada con operandos enteros desde un objeto `istream` (por ejemplo, `cin`). De esta forma se puede utilizar el método `istream::peek()` para inspeccionar si el siguiente carácter de la expresión de entrada es un operador (+ - * /), y el operador de extracción del flujo `operator<<()` para leer los operandos enteros.

Para almacenar los operandos, reutilizar la implementación de la clase Pila con `[TDATO = int]` desarrollada en la práctica 1.

Una vez comprobado el correcto funcionamiento de la calculadora con operandos enteros, se procederá a generalizar la implementación de la función calculadora para que resuelva expresiones aritméticas con operando de otros tipos de números. Para ello se implementá la función calculadora mediante una plantilla de función:

```
template<class TDATO> TDATO calculadora(istream& expresion);
```

También será necesario implementar la clase Pila de forma genérica para que admita datos del tipo requerido por la función calculadora.

```
template<class TDATO> class Pila;
```

Además de los tipos básicos del lenguaje (int, float), la calculadora debe funcionar con expresiones de operandos en los tipos de números implementados por el usuario:

- **Racional** (designado por $\mathbb{Q}$) es todo número que pueda representarse como una fracción de dos números, el numerador entero y el denominador entero positivo.

$$\mathbb{Q}=\{\frac{p}{q} | p\in\mathbb{Z}, q\in\mathbb{Z}; q>0 \wedge mcd(|p|, q)=1\}$$

  La implementación del tipo de dato Racional debe asegurar que se representa con la fracción más simple posible, esto es, que no exista un divisor común para el numerador y denominador.

- **Complejo** (designado por $\mathbb{C}$) son una extensión de los números reales que además incluyen todas las raíces de los polinomios. Todo número complejo puede representarse como la suma de un número real y un número imaginario.

Para todos los tipos de números se deben sobrecargar los métodos constructores, los operadores aritmético, y los operadores de inserción y extracción de flujo (lectura y escritura estándar). Realizar los códigos de prueba para todas las operaciones de los tipos de números implementados.

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

4. Referencias

[1] https://es.wikipedia.org/wiki/Notación_polaca_inversa