Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Práctica 2

Tecnología de Programación

Grado en Ingeniería Informática

Objetivos

- Aprovechar los mecanismos que proporciona la herencia para maximizar la reutilización de código.
- Diseñar jerarquías de herencia que permitan polimorfismo por inclusión.
- Utilizar algunos de los contenedores estándar de C++ y Java.

Elige un lenguaje orientado a objetos (C++ o Java). Realizarás toda la práctica en dicho lenguaje.

1. Expresiones matemáticas

Para implementar la máquina virtual que va a ejecutar un nuevo lenguaje de programación se necesita la posibilidad de almacenar **expresiones matemáticas** de tipo real, y realizar ciertas operaciones con ellas. Las expresiones pueden utilizar constantes y nombres de variable (ambas de tipo real), y los operadores de suma, resta, multiplicación y división. Por ejemplo:

$$3 + 2 \times 5 \tag{1}$$
$$(3 + 2) \times 5 \tag{2}$$
$$3 + 2 \times a \tag{3}$$
$$(3 + a) \times 5 \tag{4}$$

Para evaluar una expresión, es necesario conocer el valor de las variables que se utlicen en ella, que están almacenadas mediante una **tabla de símbolos** que contiene parejas `{nombre,valor}`.
Por ejemplo, con la siguiente tabla de símbolos que define que el valor de `'a'` es `7.0`:

Tabla de símbolos

a	7.0
b	1.2
pi	3.1415
long	12.5

$$3 + 2 \times 5 = 13$$
$$(3 + 2) \times 5 = 25$$
$$3 + 2 \times a = 17$$
$$(3 + a) \times 5 = 50$$

Una **expresión** matemática se puede representar mediante un **arbol sintáctico** de términos, donde cada nodo no terminal representa una operación (binaria en este caso), y cada nodo terminal es un valor constante o una variable. La evaluación de una expresión representada de esa forma puede hacerse fácilmente de forma recursiva (ver la figura 1).

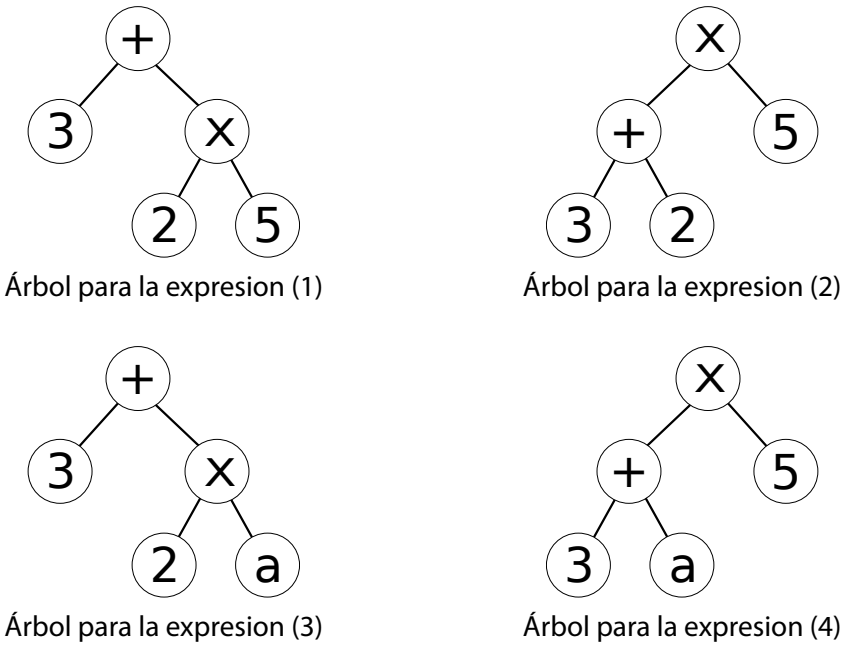

Figura 1: Árboles sintácticos para las expresiones de ejemplo.

Para comprobar que un árbol es correcto, se quiere poder imprimir por pantalla la expresión en la notación habitual, teniendo en cuenta que dependiendo de la precedencia estándar de los operadores aritméticos, pueden ser necesarios paréntesis para que la operación esté correctamente escrita:

- Dentro de el mismo nivel de precedecia (por ejemplo varias sumas concatenadas), el orden normal de evaluación de las operaciones es de **izquerda a derecha**.
- La **precedencia de las operaciones** es la siguiente: primero se evalúan las multiplicaciones y divisiones, y después las sumas y las restas.

Para facilitar el interfaz con el analizador sintáctico del lenguaje, se quiere también poder construir una expresión a partir de la notación **RPN**[1] (notación polaca inversa) para la expresión:

```
Expression e("3 2 5 * +"); // Expresion 1
Expression e("3 2 + 5 *"); // Expresion 2
Expression e("3 2 a * +"); // Expresion 3
Expression e("3 a + 5 *"); // Expresion 4
```

La construcción de la expresión puede hacerse mediante una pila de términos con la que se va operando según se leen de forma secuencial de la expresión. Por ejemplo, para la expresión 3 del caso anterior:

- `'3'`: apilar un término "`constante 3`"
- `'2'`: apilar un término "`constante 2`"
- `'a'`: apilar un término "`variable 'a'`"

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

- `'*'`: desapilar "`constante 2`" y "`variable 'a'`" y
  apilar un término "`producto(constante 2,variable 'a')`"
- `'+'`: desapilar "constante 3" y "`producto(constante 2,variable 'a')`" y
  aplilar "`suma(constante 3,producto(constante 2,variable 'a'))`"

Al terminar de leer la expresión, la cima de la pila es la raíz del árbol que representa la expresión.

2. Tareas

En un lenguaje orientado a objetos a tu elección (C++ o Java) lleva a cabo las siguientes tareas:

- Define las clases necesarias para almacenar en memoria una **expresión**, mediante un árbol de **términos**.
- Define un método que devuelva la representación textual de la expresión en la notación habitual (infija): `to_string()` en C++ o `toString()` en Java, que se usan en los operadores de escritura en ambos lenguajes.
- Añade a las expresiones un método `eval(...)`, que evalue la expresión, devolviendo un número real, y que recibe como parámetro una tabla de símbolos con los valores de las variables. Para ello necesitarás ver cómo usar el tipo `SymbolTab` (que te damos ya implementado), que almacena los valores de las variables.
- Completa el método `parse(...)` que recibe una cadena con la representación RPN de la expresión y genera el árbol correspondiente. A partir de dicho método, implementa el constructor de una expresión a partir de una cadena RPN.

Verifica el comportamiento de tu código con el programa principal suministrado en la práctica, que también debes completar.

Entrega

Los archivos de código fuente de la práctica deberán estar organizados en un subdirectorio `'c++'` o `'java'` (dependiendo del lenguaje que hayas elegido), con la siguiente estructura:

```
practica2_XXXXXX_XXXXX
     \---- c++
          \---- Makefile
          \---- main.cc
          \---- expression.h
          \---- expression.cc
          \---- ...

practica2_XXXXXX_XXXXX
     \---- java
          \---- Main.java
          \---- Expression.java
          \---- ...
```

EL directorio debe contener todos los archivos necesarios para su compilación (puedes haber añadido alguno más aparte de los que te damos).

El programa en C++ deberá ser compilable y ejecutable con los archivos que has entregado en la subcarpeta `c++` mediante los siguientes comandos:

```
make
./main
```

El programa en Java deberá ser compilable y ejecutable con los archivos que has entregado en la subcarpeta `java` mediante los siguientes comandos:

```
javac Main.java
java Main
```

En caso de no compilar siguiendo estas instrucciones, el resultado de la evaluación de la práctica será de 0. No se deben utilizar paquetes ni librerías ni ninguna infraestructura adicional fuera de las librerías estándar de los propios lenguajes.

Todos los archivos de código fuente solicitados en este guión deberán ser comprimidos en un único archivo `ZIP` con el siguiente nombre:

- `practica2_<nip>.zip` (donde `<nip>` es el NIP de 6 dígitos del estudiante involucrado) si el trabajo ha sido realizado de forma individual.
- `practica2_<nip1>_<nip2>.zip` (donde `<nip1>` y `<nip2>` son los NIPs de 6 dígitos de los estudiantes involucrados) si el trabajo ha sido realizado en pareja. En este caso **sólo uno** de los dos estudiantes deberá hacer la entrega.

El archivo comprimido a entregar no debe contener ningún fichero aparte de los fuentes que te pedimos: ningún fichero ejecutable o de objeto, ni ningún otro fichero adicional. La entrega se hará en la tarea correspondiente a través de la plataforma Moodle.