UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Evaluación Continua 1 - Prueba 3 - Tipo 3

1. Objetivo

El objetivo es diseñar e implementar un *visitor* para un lenguaje imperativo sencillo, basado en la gramática propuesta.

2. Indicaciones

Se cuenta con un *scanner*, *parser* y *visitor* que trabajan sobre la siguiente gramática:

$$
\begin{aligned}
Program &::= StmtList \\
StmtList &::= Stmt\ (';'\ Stmt)^* \\
Stmt &::= Id\ '='\ CExp \mid \text{print}\ '('\ CExp\ ')' \\
CExp &::= Expr\ \{(+ \mid -)\ Expr\}^* \\
Expr &::= Term\ \{(* \mid /)\ Term\}^* \\
Term &::= Factor\ [**\ Factor] \\
Factor &::= Number \mid (CExp) \mid \text{sqrt}(CExp) \mid Id
\end{aligned}
$$

Esta gramática define un mini-lenguaje imperativo que permite construir programas compuestos por una lista de sentencias separadas por punto y coma. Dichas sentencias pueden ser asignaciones de valores a variables mediante expresiones aritméticas o instrucciones de salida con `print`. Ejemplos de programas válidos:

1. `x = 5; y = 10; print(x + y)`
2. `a = 2 ** 3; b = sqrt(16); print(a * b)`
3. `m = (3 + 7) * 2; n = m - 5; print(n)`
4. `u = 100 / 4; v = u + 6; w = v ** 2; print(w)`
5. `x = sqrt(25); y = (x + 3) * 2; print(y - 1)`
6. `a = 7; b = 8; c = (a + b) / 3; print(c)`
7. `r = 2; s = r ** 4; t = sqrt(s); print(t)`
8. `p = 9; q = p - 4; r = q * (2 + 3); print(r)`
9. `x = 1; y = 2; z = (x + y) ** 2 - sqrt(9); print(z)`
10. `a = 2; b = 3; c = a * b + (a ** b); d = sqrt(c); print(d)`

La gramática debe ser extendida de modo que el *scanner*, el AST, el *parser* y el *visitor* soporten las siguientes construcciones adicionales:

$$
\begin{aligned}
Program &::= StmtList \\
StmtList &::= Stmt\ (';'\ Stmt)^* \\
Stmt &::= Id\ =\ CExp\ \mid\ \text{print}\ '('\ PrintArg\{,PrintArg\}^*\ ')' \\
CExp &::= Expr\ \{(+\mid -)\ Expr\}^* \\
Expr &::= Term\ \{(*\mid /)\ Term\}^* \\
Term &::= Factor\ [**\ Factor] \\
Factor &::= Number \\
&\quad \mid (CExp) \\
&\quad \mid \text{sqrt}(CExp) \\
&\quad \mid Id \\
&\quad \mid min\ '('\ Cexp\ \{,\ Cexp\}^*\ ')' \\
&\quad \mid rand\ '('\ Cexp,\ Cexp\ ')' \\
PrintArg &::= CExp\ \mid\ String \\
String &::= '\ id\ '
\end{aligned}
$$

Ejemplos de cadenas válidas con las nuevas reglas:

1. `x = 5; print(x)`
2. `z = (3 + 5) * 2; print('z es', z)`
3. `y = sqrt(16); print('Resultado de y:', y)`
4. `a = min(3, 7); print(a)`
5. `b = rand(1, 10); print('Número aleatorio:', b)`
6. `m = 10; n = 4; print('m-n:', m-n)`
7. `p = min(8, 6, 9, 2); print('mínimo:', p)`
8. `r = rand(5, 15) + min(4, 9); print('valor r:', r)`
9. `x = sqrt(25); y = rand(1,5); print('x=', x, ' y=', y)`
10. `val = min( rand(1,100), sqrt(81), 50 ); print('Resultado final:', val)`

3. Sugerencias

1. **Modificar el scanner:** Incluir el reconocimiento de los siguientes tokens:

Token	Patrón
MIN	`min`
STRING	`string`
COMA	`,`
RAND	`rand`

2. **Extender el AST:** Implementar las clases `MinExp`, `RandExp` para representar las nuevas expresiones.

3. **Actualizar el parser y el visitor:** Ajustar las reglas del parser para reconocer los nuevos operadores y funciones, asegurando su correcta precedencia. Extender el *visitor* para evaluar o recorrer las expresiones de manera coherente con la semántica del lenguaje.