

GRAMATICA:

A continuación, se procederá a explicar la gramática utilizada para completar los requisitos del proyecto. Primero echemos un vistazo a las palabras que componen a la gramática y su significado:

ONE_LINE_COMMENT: Hace referencia al comentario de una línea.

MULTI_LINE_COMMENT: Hace referencia al comentario multilínea.

CONCATENATION: Hace referencia a la concatenación dentro de la expresión regular.

DISJUNCTION: Hace referencia a la disyunción dentro de la expresión regular.

NO_OR_MORE: Hace referencia a la cerradura de Kleene dentro de la expresión regular.

ONE_OR_MORE: Hace referencia a "+" dentro de la expresión regular.

NO_OR_ONE: Hace referencia a "?" dentro de la expresión regular.

SET: Forma la palabra reservada "CONJ" que a su vez es la que nos permite crear conjuntos.

LOWER: Se refiere al carácter que tiene letra minúscula.

UPPER: Se refiere al carácter que tiene letra mayúscula.

NUMBERS: Se refiere realmente solo a un número.

COMMA: Se refiere a la separación por comas ",".

SEPARATOR: Se refiere al separador circunflejo "~".

NEW_LINE: Se usa para que no colapse el programa y es un salto de línea.

TAB: Se usa para que no colapse el programa y es una tabulación.

SPACE: Se usa para que no colapse el programa y es un espacio simple.

CAR_RETURN: Se usa para que no colapse el programa y es un retorno de carro.

BEGIN: Se refiere al corchete realmente, cada vez que se mencione se refiere al corchete abierto "{".

END: Se refiere al corchete realmente, cada vez que se mencione se refiere al corchete cerrado "}".

PORCENTAGE: Se usa para realizar la separación del set de instrucciones y es un porcentaje "%".

COLON: Son los dos puntos y nos sirve para realizar conjuntos o para definir las expresiones regulares.

SEMI_COLON: Es el punto y coma, nos sirve para realizar conjuntos o para definir las expresiones finalizar una línea con instrucción.

CHARACTER: Es un carácter dentro de un comentario.

TEXT : Es un arreglo de caracteres dentro de un comentario.

SPECIAL: Se refiere a todos los caracteres distintos a números y letras que comprenden el código ASCII del 32 al 125.

VARIABLE: Son las variables que van a “guardar” el nombre de la expresión.

SLASH: Realmente es el guión bajo: “_”.

MAYOR: Es el carácter que representa al mayor: >.

Con todas las palabras ya estudiadas, ya podemos pasar a llamarles como TERMINALES. Veamos con qué inicia nuestra gramática:

Esta inicia con el no terminal “prueba”

```
prueba::= BEGIN first_set PORCENTAGE PORCENTAGE PORCENTAGE PORCENTAGE second_set
END;
```

Análisis: El “BEGIN” marca el inicio del programa como tal ya que es el corchete abierto, si viene un comentario de cualquier tipo, no tendremos problema ya que cuando uno de estos es detectado es automáticamente ignorado.

Luego vamos con el no terminal “first_set” y se refiere al primer conjunto de reglas o instrucciones a ejecutar, luego le siguen 4 porcentajes que es lo que marca el inicio del segundo no terminal “second_set”, el cual representa al segundo grupo de instrucciones, por último con “END” le decimos al programa que se detenga.

```
{ BEGIN
///// CONJUNTOS      Comentarios son ignorados por la
                      gramática
CONJ: letra -> a~z;
CONJ: digito -> 0~9;

///// EXPRESIONES REGULARES
                      El primer set de instrucciones
ExpReg1 -> . {letra} * | "_" | {letra} {digito};
ExpresionReg2 -> . {digito} . "." + {digito};
RegEx3 -> . {digito} * | "_" | {letra} {digito};

%%
%%                      Los porcentajes

ExpReg1 : "primerLexemaCokoa";
ExpresionReg2 : "34.44";
} END                      El segundo set de instrucciones
```

Como se puede apreciar, en este momento no hemos necesitado aplicar la recursividad ya que no es necesario porque el orden de la estructura es lineal, ya para nuestro primer set de instrucciones y también para el segundo ya se utilizó la recursividad por la izquierda.

```
first_set ::= first_set new_instruction |
```

```
new_instruction;
```

El no terminal new_instruction, será el encargado de romper la recursividad y este esta dado de la siguiente forma:

```
new_instruction ::= SET COLON VARIABLE SLASH MAYOR patron SEMI_COLON |
```

```
VARIABLE SLASH MAYOR regex SEMI_COLON;
```

La primera línea es para que se sepa que se trata de un conjunto, y este tiene la siguiente forma:



```
CONJ: letra -> a~z;  
CONJ: digito -> 0~9;
```

Con el patron es que se va a visualizar como funciona:

```
patron ::= NUMBERS SLASH NUMBERS |
```

```
LOWER SEPARATOR LOWER |
```

```
UPPER SEPARATOR UPPER |
```

```
SPECIAL SEPARATOR SPECIAL |
```

```
coma_notation;
```

Con este se refiere a: “~”, con el no terminal de “coma_notation” es que se cubre con las comas:

```
coma_notation ::= coma_notation COMMA simple |
```

```
simple;
```

En su lugar de utilizar un patrón, debido a que necesitamos solo agregar uno, se hizo el no terminal simple, el cual tiene la siguiente forma:

```
simple ::= NUMBERS |
```

```
LOWER |
```

```
UPPER |
```

SPECIAL;

Ya habiendo cubierto la forma en la que se trabaja la primera parte del set de instrucciones, esta tiene otra forma de asignar y es la siguiente:

VARIABLE SLASH MAYOR regex SEMI_COLON;

El cual ya trabaja con la estructura del regex, el cual es un no terminal con la siguiente forma:

```
regex::= CONCATENATION:a regex:b regex:c{:
    node parent_node = new node(a, "AND");
    parent_node.left = (node)b;
    parent_node.right = (node)c;
    RESULT = parent_node;
:}|
DISJUNCTION:a regex:b regex:c{:
    node parent_node = new node(a, "OR");
    parent_node.left = (node)b;
    parent_node.right = (node)c;
    RESULT = parent_node;
:}|
NO_OR_MORE:a regex:b{:
    node parent_node = new node(a, "KLEENE");
    parent_node.left = (node)b;
    RESULT = parent_node;
:}|
ONE_OR_MORE:a regex:b{:
    node parent_node = new node(a, "SUMA");
    parent_node.left = (node)b;
    RESULT = parent_node;
:}|
NO_OR_ONE:a regex:b{:
    node parent_node = new node(a, "INTE");
    parent_node.left = (node)b;
    RESULT = parent_node;
:}|
CHARACTER:a{:
    node leaf = new node(a.charAt(1) + " ", "HOJA");
    System.out.println("HEEEEEEE RECONOCIDO: " + a.charAt(1));
    leaf.leaf = true;
    RESULT = leaf;
:}|
LOWER:a{:
    node leaf = new node(a + " ", "HOJA");
    leaf.leaf = true;
    RESULT = leaf;
:}|
UPPER:a{:
    node leaf = new node(a, "HOJA");
```

```
leaf.leaf = true;  
RESULT = leaf;  
:}|  
PARENTESIS_ABIERTO VARIABLE:a PARENTESIS_CERRADO {:  
node leaf = new node(a, "HOJA");  
leaf.leaf = true;  
RESULT = leaf;  
:};
```

Tanto como “*”, “+” y “?”, solo tienen un no terminal de regex al lado debido a que estos no requieren su agrupación en pares, este es recursivo por la derecha.

Al final tenemos un carácter, Lower Upper y la ultima expresión los cuales funcionan como nodos hoja y son identificados para poder realizar las operaciones de manera correcta.