Modelado de expresiones regulares

A continuación se modela un grupo de expresiones regulares para poder identificar un patrón de comportamiento de sus respectivos tokens.

Empezando por los identificadores, estos empiezan por una letra o un guión bajo seguido de al menos un digito o una letra, a ambas posibilidades le puede seguir cero o muchas veces un guion bajo, una letra o un numero.

```
Identificadores ((\_+([a-zA-Z]|[0-9])+|[a-zA-Z])(\_|[a-zA-Z]|[0-9])*)
```

El decimal puede o no traer un guion medio que en es interpretado como un signo de negatividad, seguido de cualquier digito entre uno y nueve seguido de ninguno o varios digitos entre cero y nueve, sinó, un solo cero; continuando con un punto y al menos un digito entre cero y nueve.

```
Decimal (((-?[1-9][0-9]*)|0)(\.([0-9]+)))
```

El entero puede o no traer un guion medio que en es interpretado como un signo de negatividad, seguido de cualquier digito entre uno y nueve seguido de ningun o varios digitos entre cero y nueve; sinó, un solo cero.

```
Enteros
((-?[1-9][0-9]*)|0)
```

Las cadenas aceptan una comilla doble o simple, seguido de cualquier carácter que no sea una secuencia de escape, ya sea nueva linea o retorno de carro. Finaliza con otra comilla simple o doble.

```
Cadenas
("([^"'\n\\]|\\.)*"|'([^'"\n\\]|\\.)*')
```

Los comentarios reciben un simbolo de numeral, seguido de cualquier carácter que no sea las secuencias de escape de retorno de carro o nueva linea con los cuales termina el patrón.

```
Comentarios (#([^\r\n]*)\n)
```

Los aritmeticos son una secuencia de Or's para cada uno de ellos.

```
Aritmeticos
(\+|\-|\*\*|\*|\/\/|\/|\%)
```

Para comparacion se aplica la misma idea que en los aritmeticos.

```
Comparacion (\=\=|\!\=|\<\=|\>\=|\<|\>)
```

Asignación repite lo mismo de los dos casos anteriores.

```
Asignacion (\+\=|\-\=|\*\*=|\/\/\=|\%\=|\=)
```

Y es 'otros' quienes finalizan este procedimiento llenos de or's

```
(\(|\)|\{|\}|\[|\]|\,,|\;|\:)
```

Las palabras claves para no tener que hacer lo mismo que con los ultimos cuatro casos, se aprovechó que estas cumplen la expresion de los identificadores por lo que para hacer mas escalable el desarrollo, cuando se vaya a realizar el registro de un identificador este se verificará si está en la lista de palabras reservadas, si es el caso se hacen los cambios correspondientes y se guarda el token.

Con los numeros decimales y enteros se notó que si en el patron de decimales se hace opcional la parte correspondiente al punto decimal y los digitos que le siguen, este tambien acepta los numeros enteros, razón por lo que se tomó la desición de omitir el patron para enterode manera individual y el patron unificado para numeros enteros y numeros decimales quedaria de la siguiente manera:

```
Enteros y decimales
(((-?[1-9][0-9]*)|0)(\.([0-9]+))?)
```

Similar situación pasará con algunos elementos de las expresiones regulares correspondientes a aritmetícos, comparación, asignación y otros; debido a que son casi en su totalidad estos están compuestos de caracteres inividuales, por lo que pueden ser listados y si están dentro de dicha lista serán guardados de forma automatica.

La expresión regular sin las optimizaciónes seria:

```
Pero, la expresion reducida para optimizar la escalabilidad y lectura del codigo seria:
```

Tabla de transiciónes:

Min-DFA STATE	TYPE	!		#	%,<,=,>,t		,,:,;,[,],o,u,{,}	-		1	0	=	D	_	_,d,w	С	d	d,w	n	s	w	x
1		11	3	4	6	5	2	13		14	15		8	9							10	16
2	accept																					
3			2																	3		
4																4			2			
5						2														5		
6	accept											2										
7																	12					
8	accept								7								8					
9														9				10				
10	accept														10							
11												2										
12	accept																12					
13	accept											2	8									
14	accept									6		2										
15	accept								7													
16	accept											2										6

