1 Expresiones Regulares Patito

Token	**Regex**
TYPE	int|float
id	$[a-zA-Z][\_a-zA-Z0-9]*$
program	program
var	var
if	if
else	else
do	do
while	while
end	end
cout	cout
cte.string	".*"
cte_int	`^\d+$`
cte_float	`^\d+\.\d+([eE][-+]?\d+)?$`
semicolon	;
coma	,
colon	:
leftBrace	{
rightBrace	}
leftParentesis	(
rightParentesis	)
equal	=
plus	+
minus	-
multiplication	*
division	/
lessThan	<
greaterThan	>
notEqual	!=

2 Reglas Gramaticales Patito

TYPE

```
TYPE -> int
TYPE -> float
```

VARS

```
VARS -> var VARS'
VARS' -> ids : TYPE ; VARS''
VARS'' -> VARS'
VARS'' -> epsilon
ids -> id ids'
ids' -> , ids
ids' -> epsilon
```

CTE

```
CTE -> cte_int
CTE -> cte_float
```

Programa

```
Programa -> program id ; variables Body end
variables -> VARS
variables -> epsilon
```

Body

```
Body -> { Body'
Body' -> STATEMENT Body'
Body' -> }
```

STATEMENT

```
STATEMENT -> ASSIGN
STATEMENT -> CONDITION
STATEMENT -> CYCLE
STATEMENT -> Print
```

ASSIGN

```
ASSIGN -> id = EXPRESION ;
```

CYCLE

```
CYCLE -> do Body while ( EXPRESION ) ;
```

Print

```
Print -> cout ( Print'
Print' -> cte.string Print''
Print' -> EXPRESION Print''
Print'' -> , Print'
Print'' -> ) ;
```

CONDITION

```
CONDITION -> if ( EXPRESION ) Body CONDITION' ;
CONDITION' -> else Body
CONDITION' -> epsilon
```

EXPRESION

```
EXPRESION -> EXP EXPRESION'
EXPRESION' -> < EXP
EXPRESION' -> > EXP
EXPRESION' -> != EXP
EXPRESION' -> epsilon
```

EXP

```
EXP -> TERMINO EXP'
EXP' -> + TERMINO EXP'
EXP' -> - TERMINO EXP'
EXP' -> epsilon
```

TERMINO

```
TERMINO -> FACTOR TERMINO'
TERMINO' -> * TERMINO
TERMINO' -> / TERMINO
TERMINO' -> epsilon
```

FACTOR

```
FACTOR -> ( EXPRESION )
FACTOR -> + FACTOR'
FACTOR -> - FACTOR'
FACTOR -> FACTOR'
```

FACTOR'-> id
FACTOR' -> CTE