



**Desarrollo de aplicaciones avanzadas de ciencias computacionales**

## Actividad Final Mini Proyecto Parte 1

**Presentado por:**

Felipe Gabriel Yépez Villacreses

A01658002

**Profesores:**

Elda Guadalupe Quiroga González

Iván Mauricio Amaya Contreras

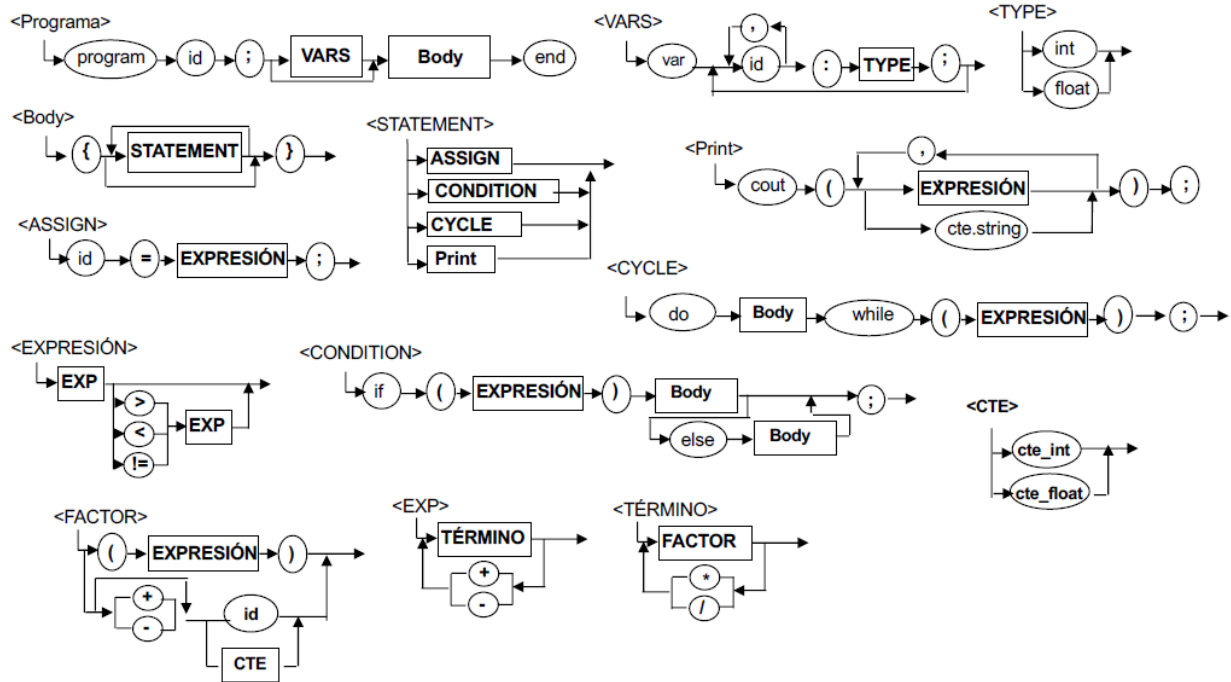
Edgar Covantes Osuna

Alexis Edmundo Gallegos Acosta

**Fecha de entrega**

Sábado 20 de mayo 2023

# Mini Proyecto: Patito



## Expresiones Regulares

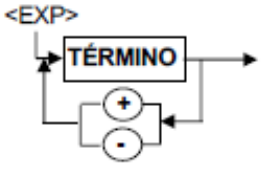
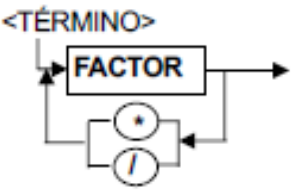
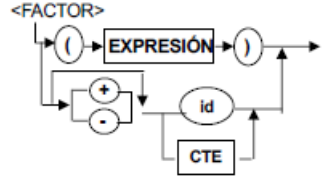
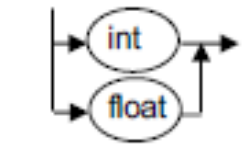
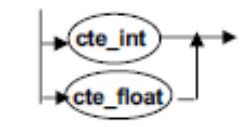
Diseño de expresiones regulares para los diferentes elementos del léxico de lenguaje Patito.

- reserved -> [ program end var int float cout if else do while]
- id -> [ a-z A-Z ] [ a-z A-Z 0-9 ]\*
- cte\_string -> ".\*?"
- cte\_int -> [ 0-9 ]+
- cte\_float -> [ 0-9 ]+ . [ 0-9 ]+
- leftParenthesis -> (
- rightParenthesis -> )
- leftBrace -> {
- rightBrace -> }
- colon -> :
- coma -> ,
- semicolon -> ;
- equal -> =
- add -> +
- minus -> -
- multiply -> \*
- divide -> /
- greaterThan -> >
- lessThan -> <

- not -> !=

## Gramáticas libres de contexto

<p>&lt;Programa&gt;</p>	<p><math>\langle \text{program} \rangle \rightarrow \text{program id ; R } \langle \text{body} \rangle \text{ end}</math>  <math>R \rightarrow \langle \text{vars} \rangle</math>  <math>R \rightarrow \epsilon</math></p>
<p>&lt;VARS&gt;</p>	<p><math>\langle \text{vars} \rangle \rightarrow \text{var } O</math>  <math>O \rightarrow \text{id } P</math>  <math>P \rightarrow , O</math>  <math>P \rightarrow : \langle \text{type} \rangle ; Q</math>  <math>Q \rightarrow \epsilon</math>  <math>Q \rightarrow O</math></p>
<p>&lt;Body&gt;</p>	<p><math>\langle \text{body} \rangle \rightarrow \{ M \}</math>  <math>M \rightarrow \langle \text{statement} \rangle M</math>  <math>M \rightarrow \epsilon</math></p>
<p>&lt;STATEMENT&gt;</p>	<p><math>\langle \text{statement} \rangle \rightarrow \langle \text{assign} \rangle</math>  <math>\langle \text{statement} \rangle \rightarrow \langle \text{condition} \rangle</math>  <math>\langle \text{statement} \rangle \rightarrow \langle \text{cycle} \rangle</math>  <math>\langle \text{statement} \rangle \rightarrow \langle \text{print} \rangle</math></p>
<p>&lt;ASSIGN&gt;</p>	<p><math>\langle \text{assign} \rangle \rightarrow \text{id} = \langle \text{expression} \rangle ;</math></p>
<p>&lt;CYCLE&gt;</p>	<p><math>\langle \text{cycle} \rangle \rightarrow \text{do } \langle \text{body} \rangle \text{ while } ( \langle \text{expression} \rangle ) ;</math></p>
<p>&lt;CONDITION&gt;</p>	<p><math>\langle \text{condition} \rangle \rightarrow \text{if } ( \langle \text{expression} \rangle ) \langle \text{body} \rangle L ;</math>  <math>L \rightarrow \epsilon</math>  <math>L \rightarrow \text{else } \langle \text{body} \rangle</math></p>
<p>&lt;EXPRESIÓN&gt;</p>	<p><math>\langle \text{expression} \rangle \rightarrow \langle \text{exp} \rangle J</math>  <math>J \rightarrow \epsilon</math>  <math>J \rightarrow K \langle \text{exp} \rangle</math>  <math>K \rightarrow &gt;</math>  <math>K \rightarrow &lt;</math>  <math>K \rightarrow !=</math></p>
<p>&lt;Print&gt;</p>	<p><math>\langle \text{print} \rangle \rightarrow \text{cout } ( G ) ;</math>  <math>G \rightarrow H I</math>  <math>H \rightarrow \langle \text{expression} \rangle</math>  <math>H \rightarrow \text{cte\_string}</math>  <math>I \rightarrow \epsilon</math>  <math>I \rightarrow , G</math></p>

	$\langle exp \rangle \rightarrow \langle term \rangle E$ $E \rightarrow \epsilon$ $E \rightarrow F \langle term \rangle$ $F \rightarrow +$ $F \rightarrow -$
	$\langle term \rangle \rightarrow \langle factor \rangle C$ $C \rightarrow \epsilon$ $C \rightarrow D \langle term \rangle$ $D \rightarrow *$ $D \rightarrow /$
	$\langle factor \rangle \rightarrow ( \langle expression \rangle )$ $\langle factor \rangle \rightarrow AB$ $A \rightarrow \epsilon$ $A \rightarrow +$ $A \rightarrow -$ $B \rightarrow id$ $B \rightarrow \langle cte \rangle$
	$\langle type \rangle \rightarrow int$ $\langle type \rangle \rightarrow float$
	$\langle cte \rangle \rightarrow cte\_int$ $\langle cte \rangle \rightarrow cte\_float$

## Funcionamiento Alcanzado

Para realizar el código utilicé la librería `PLY` en Python. Mediante el uso de `yacc` y `lex` logré plasmar lo anteriormente descrito en este documento para incluir las expresiones regulares al crear el analizador léxico y la gramática libre de contexto al crear el analizador sintáctico.

Hasta este punto de desarrollo se puede detectar tokens que no se hayan reconocido en el lenguaje Patito y mostrar qué tokens fueron. De igual forma se puede mostrar qué tipo de token se ha detectado correctamente con su respectivo identificador.

El analizador sintáctico puede determinar si un archivo completo del lenguaje Patito sigue las reglas gramaticales, caso contrario determina que hubo un error de sintaxis.

Para probar el analizador léxico se probó con lo siguiente.

### Pruebas Analizador Léxico

Prueba correcta:

```
test_lexer_valido.txt
1  Var1 "food & dr1nk5"
2  5 4.3
3  ( ) { } : ; , =
4  + - * / > < !=
5  program programa
6  end ends
7  var int float cout if else do while
```

Prueba incorrecta:

```
test_lexer_invalido.txt
1  Var1 "food & dr1nk5"
2  5 4.3
3  ( ) { } : ; , =
4  + - * / > < !=
5  program programa
6  end ends
7  var int float cout if else do while
8  ^ @ 4a ! a4
```

## Pruebas Analizador Sintáctico

Prueba correcta:

```
test_parser_valido.txt
1  program Felipe;
2
3  var num, i, j: int;
4  mean: float;
5
6  {
7      num = 0;
8      i = 5 * 4 + 3.1;
9      j = 0;
10     if(i != 5){
11         i = 5;
12     }
13     else{
14         mean = i;
15     };
16     if(5 > 4){
17
18     };
19     do{
20         j = j + 1;
21         cout ("Print");
22     } while(j < 3);
23     cout (+3.2 * mean + 4);
24     cout ((-num * 5.3 + 5) + 6 > -4 * 5, "operacion");
25     cout(5, 6);
26     cout(5.7);
27 }
28 end
```

Prueba incorrecta:

test\_parser\_invalid.txt

```
1  programa % Felipe;
2
3  var num, i, j: int;
4  mean: float;
5
6  {
7      num = 0;
8      i = 5 * 4 + 3.1;
9      j = 0;
10     if(i != 5){
11         i = 5;
12     }
13     else{
14         mean = i;
15     };
16     if(5 > 4){
17
18     };
19     do{
20         j = j + 1;
21         cout ("Print");
22     } while(j < 3);
23     cout (+3.2 * mean + 4);
24     cout ((-num * 5.3 + 5) + 6 > -4 * 5, "operacion");
25     cout(5, 6);
26     cout(5.7);
27 }
28 end
```

## Salidas Obtenidas

```
to>python Scanner_Parser_Patito.py
Generating LALR tables
Testing incorrect lexer...
  Invalid character:  ^
  Invalid character:  @
  Invalid character:  !

Testing correct lexer...

Testing correct parser...

Testing incorrect parser...
  Syntax error in input:
  Invalid character:  %
```

Se logra detectar léxicos incorrectos cuando los hay y también se detecta cuando el archivo no logra compilar por errores de sintaxis.