Presentación Gramática

DOCENTE	CARRERA	CURSO
MSc. Vicente Enrique Machaca Arceda	Escuela Profesional de Ingeniería de Software	Compiladores

1. Datos de los estudiantes

- Grupo: 1
- Git Hub: `https://github.com/CrazyDani17/GramaticaVersionTokens`
- Integrantes:
  - Guillermo Aleman
  - Marvik Del Carpio
  - Daniel Mendiguri
  - Daniela Vilchez

2. Preguntas

1. Informe: Elaborar un documento que describa el lenguaje propuesto. El documento debe contener lo siguiente:
   - Introducción:

     La programación esta inmersa en todo, y cada vez se hace más necesario tener idea de qué es, o cómo se trabaja, para ello en muchas currículas de formación básica se incluyen cursos referidos, y las asignaturas de computación dejaron de tener un enfoque específicamente ofimático.
     Ante la necesidad de que el programar esté al alcance de los que no son especialistas ni apuntan a ello, pero requieren entender cómo se hace; y además para aquellos que buscan despertar el interés en este campo nace Llama.
     La idea de este proyecto se origina en el Curso de Compiladores de la Universidad La Salle, con el fin de encontrar una forma de integrar a este campo especialmente a niños y adolescentes a través de despertar su interés por medio de lo simple, para ello nos concentramos en lograr lenguaje sencillo de entender para latino-hablantes, uno sin exceso de reglas que se concentrase en dar la lógica con la sintaxis y estructuras de control elementales.
     Para poder definir el nombre, se observó la particularidad de lenguajes como python, herramientas de programación como anaconda, spyder o mysql. Todas ellas refirieren a un animal; entonces rápidamente se pensó en uno característico de Perú, y que además pueda tener sentido respecto a la operabilidad del lenguaje.
     Llama es el nombre definido, y tomó un mayor sentido cuando se inició la construcción del lenguaje. Llama se desarrolla a partir de funciones secundarias las cuales pueden ejecutar sentencias e interactuar entre sí, para luego ser llamadas finalmente por la función principal "llama".

- Lenguaje de programación LLAMA

- Especificación léxica: Describa cada token y muestre las expresiones regulares.

  a) Definición de los comentarios.
  Comentarios en bloque: ∼texto∼

  b) Definición de los identificadores. Los identificadores deben empezar con una letra, no está permitido que un identificador empiece con un número o un subguión.
  Permite:
  - Los caracteres de la "A" a la "Z".
  - Los caracteres de la "a" a la "z".
  - Números y subguión.

  c) Definición de las palabras clave.
  si (if)
  sino (else)
  pinocho (bool)
  numero (int)
  decimal (double)
  mostrar (print)
  texto (string)
  descansito (break)
  yoyo (while)
  devuelve (return)
  chacha (char)

  d) Definición de los literales.
  - Literales enteros
    1
  - Literales entero flotante
    2.0
  - Literales booleanos
    verdad, mentira
  - Literales caracteres

    ```
    nueva linea \n
    tabulador \t
    ```

- Literales string

```
string vacio ""
string "asdasda"
```

e) Definición de los operadores. Los siguientes caracteres se utilizan en el código fuente como operadores:

```
+ - * / & %
```

Las siguientes combinaciones de caracteres se utilizan como operadores:

```
== <= >= <> #y #o
```

f) Expresión regular de cada componente léxico (en una tabla).

Componente Léxico	**Expresión Regular**
Comentario en Bloque	\ ∼ (\w |\W)*\∼
Identificadores	[a-zA-Z]+[\w]*
String Vacio	""
Literal Entero	0|[1-9][0-9]*
Literal Flotante	(0|[1-9][0-9]*)\.[0-9]*
Literal Booleano	verdad|mentira
Nueva Linea	\n
Tabulador	\t
numero	"*numero*"
texto	"*texto*"
decimal	"*decimal*"
pinocho	"*pinocho*"
chacha	"*chacha*"
mision	"*mision*"
si	"*si*"
yoyo	"*yoyo*"
verdad	"*verdad*"
mentira	"*mentira*"
devuelve	"*devuelve*"
sino	"*sino*"
+	" + "
-	" − "
*	"*"
#*y*	"#*y*"
#*o*	"#*o*"
/	"/"
>	">"
<	"<"
(	"("
)	")"
[	"["
]	"]"
,	" , "
.	" . "
=	" = "
<=	"<="
>=	">="
<>	"<>"
%	" %"

- Gramática: Muestre la gramática. Para comprobar si la gramática está bien, puede utilizar esta herramienta (la gramática no debe ser ambigua y debe estar factorizada por la izquierda).

```
''' 
Inicio →E FuncionPrincipal K
Inicio →FuncionPrincipal
E → DeclaracionFuncion E'
E' → DeclaracionFuncion E'
E' → ''
DeclaracionFuncion →mision identificador pizquierdo Parametros pderecho
    lizquierdo CuerpoF lderecho
FuncionPrincipal →llama pizquierdo pderecho lizquierdo Cuerpo lderecho
K → ''
K → E
Parametros →''
Parametros →Y Y'
Y' → , Y Y'
Y' → ''
Y → TipoDato identificador C
TipoDato →pinocho
TipoDato →numero
TipoDato →decimal
TipoDato →texto
TipoDato →chacha
C → cizquierdo cderecho
C → ''
CuerpoF →Cuerpo J
J → ''
J → devuelve Expresion
Cuerpo →''
Cuerpo →DeclaracionVariable D'
Cuerpo →Sentencias D'
D' → DeclaracionVariable D'
D' → Sentencias D'
D' → ''
DeclaracionVariables →DeclaracionVariable Dc'
Dc' → DeclaracionVariable Dc'
Dc' → ''
Sentencias →lizquierdo MuchasSentencias lderecho
Sentencias →si pizquierdo Expresion pderecho Sentencias sino Sentencias
Sentencias →yoyo pizquierdo Expresion pderecho lizquierdo MuchasSentenciasYoyo
    lderecho
Sentencias →mostrar pizquierdo Expresion pderecho
Sentencias →identificador OPS
OPS → = Expresion
OPS →  cizquierdo Expresion cderecho igual Expresion
OPS →  pizquierdo ParaLLamados pderecho

MuchasSentenciasYoyo →SentenciasYoyo M'
M' → SentenciasYoyo M'
M' → ''
MuchasSentenciasYoyo →''

SentenciasYoyo →lizquierdo MuchasSentenciasYoyo lderecho
SentenciasYoyo →si pizquierdo Expresion pderecho SentenciasYoyo sino
```

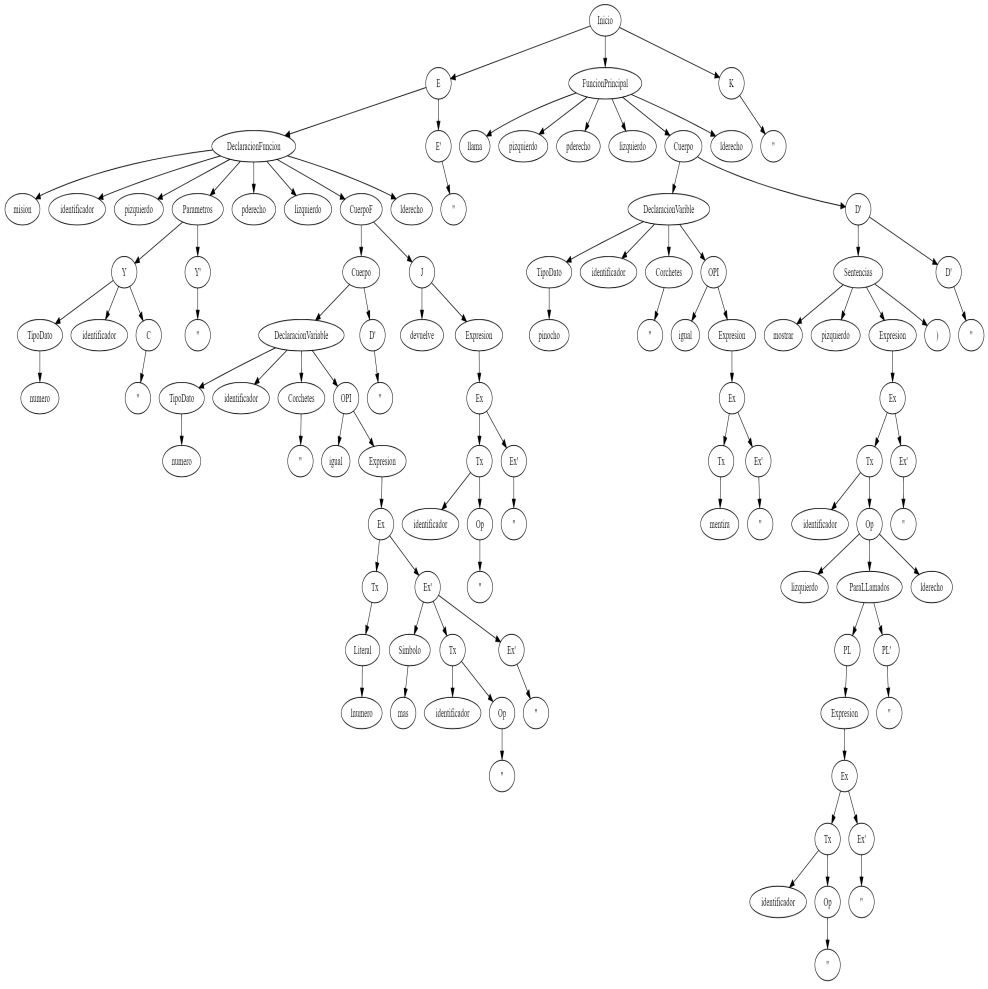
```
    SentenciasYoyo
SentenciasYoyo →yoyo pizquierdo Expresion pderecho lizquierdo
    MuchasSentenciasYoyo lderecho
SentenciasYoyo →mostrar pizquierdo Expresion pderecho
SentenciasYoyo →identificador OPS
SentenciasYoyo →descansito

MuchasSentencias →Sentencias S'
S' → Sentencias S'
S' → ''
DeclaracionVariable →TipoDato identificador Corchetes OPI
Corchetes →cizquierdo Expresion cderecho
Corchetes →''
OPI → igual Expresion
OPI → ''
Expresion →Ex
Ex → Tx Ex'
Ex' → Simbolo Tx Ex'
Ex' → ''
Tx → Literal
Literal →lpinocho
Literal →lnumero
Literal →ltexto
Literal →ldecimal
Literal →lchacha
Tx → identificador Op
Op → ''
Op → cizquierdo Expresion cderecho
Op → pizquierdo ParaLLamados pderecho
ParaLLamados →''
ParaLLamados →PL PL'
PL' → , PL PL'
PL' → ''
PL → Expresion
Tx → verdad
Tx → mentira
Tx → pizquierdo Expresion pderecho
Simbolo → mas
Simbolo → menos
Simbolo → por
Simbolo → o
Simbolo → y
Simbolo → mayor_igual
Simbolo → diferente
Simbolo → menor_igual
Simbolo → comparacion
Simbolo → menor
Simbolo → mayor
Simbolo → modulo
Simbolo → dividir
Simbolo → igual


'''
```

- Ejemplos de código: Incluya 3 ejemplos de código del lenguaje propuesto, también deberá generar el árbol sintáctico de cada ejemplo. Para graficar el árbol sintáctico use Graphviz.

```
mision identificador ( numero identificador ) {
    numero identificador = literal + identificador
    devuelve identificador
}
llama () {
    pinocho identificador = mentira
    mostrar ( identificador ( identificador ) )
}
```

Tokens generados por el analizador léxico

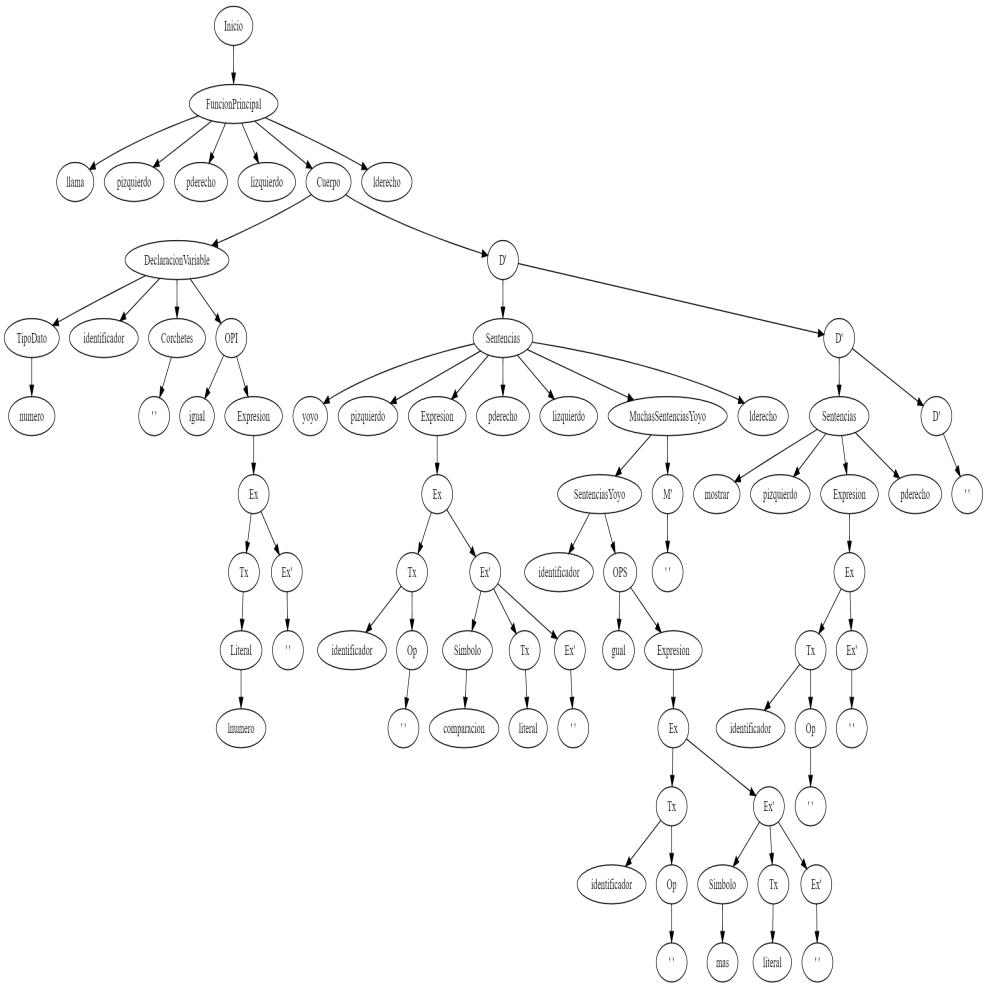
```
mision identificador pizquierdo numero identificador pderecho lizquierdo
     numero identificador igual lnumero mas identificador devuelve
     identificador lderecho llama pizquierdo pderecho lizquierdo pinocho
     identificador igual mentira mostrar pizquierdo identificador pizquierdo
     identificador pderecho pderecho lderecho
```



```
llama ()
{
    numero identificador = literal
    yoyo ( identificador == literal ) {
        identificador = identificador + literal
    }
    mostrar ( identificador )
}
```

Tokens generados por el analizador léxico

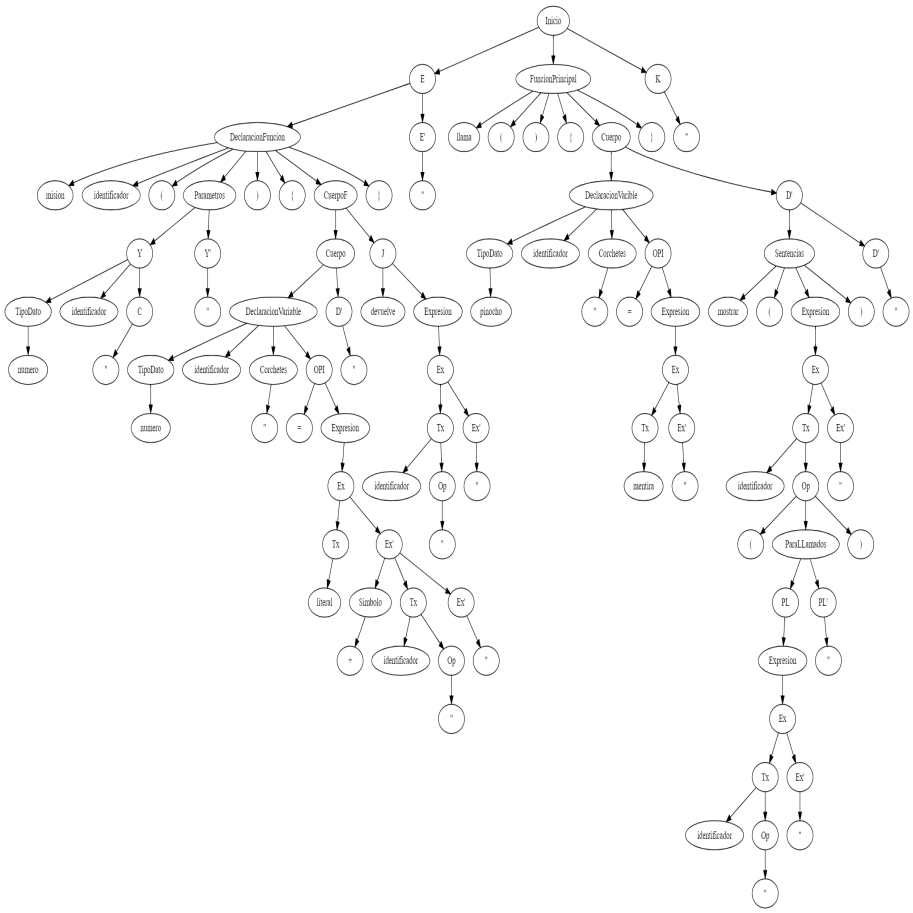
```
llama pizquierdo pderecho lizquierdo numero identificador igual lnumero
    yoyo pizquierdo identificador comparacion lnumero pderecho lizquierdo
    identificador igual identificador mas lnumero lderecho mostrar
    pizquierdo identificador pderecho lderecho
```



```
llama ()
{
    numero identificador = literal
    yoyo ( identificador < literal ) {
        si ( identificador % literal ) {
            identificador = identificador + literal
        }
        sino { descansito }
    }
    mostrar ( identificador )
}
```

Tokens generados por el analizador léxico

```
llama pizquierdo pderecho lizquierdo numero identificador igual lnumero
    yoyo pizquierdo identificador menor lnumero pderecho lizquierdo si
    pizquierdo identificador modulo lnumero pderecho lizquierdo
    identificador igual identificador mas lnumero lderecho sino lizquierdo
    descansito lderecho lderecho mostrar pizquierdo identificador pderecho
    lderecho
```



- Conclusiones

  - La implementación de un lenguaje es una tarea compleja en el sentido de la cantidad de trabajo que implica, empero sencilla por la mecanización de la lógica una vez que ésta queda definida.

  - El desarrollo de la especificación léxica debe considerar el nivel de sencillez que se busque para el lenguaje, evitando especialmente términos que sean muy parecidos, de forma tal que cada palabra reservada represente algo de forma exclusiva, esto se explica en el hecho de tener una sola clase de ciclo, por ejemplo:

  - gramática puede parecer una tarea sencilla, sin embargo, implica encajar correctamente las reglas para encajar el uso de cada término definido en la especificación léxica, esto quiere decir dar sentido real al lenguaje propuesto, para que pueda ser usado.

  - El lenguaje propuesto es simple, y contribuye a que otros puedan entender fácilmente los preceptos de la programación. Además, su construcción aportó a la práctica y uso de los conocimientos adquiridos en el curso de Compiladores.