



**UNIVERSIDAD  
TECNOLÓGICA NACIONAL**  
FACULTAD REGIONAL  
RESISTENCIA



**Trabajo Práctico Integrador:** Intérprete de Pseudocódigo en Español

Salomón, Hilel Mauricio

Segnana, Juan Franco

Zelinka, Gonzalo

Sintaxis y Semánticas de Lenguajes

Ingeniería en Sistemas de Información

Primer Cuatrimestre 2021

Universidad Tecnológica Nacional - Facultad Regional de Resistencia

Entrega 1: 25/04/21

Entrega 2: 30/05/21

Entrega 3: 04/07/21

Versión 3.0

# Indice

Introducción .....	3
Lenguaje de pseudocódigo .....	3
<b>Componentes léxicos</b> .....	3
Palabras reservadas .....	3
Identificadores .....	3
Tipos de Datos .....	4
<b>Sentencias</b> .....	5
Gramática .....	7
<b>Descripción de la gramática</b> .....	7
<b>Producciones</b> .....	8
Análisis léxico .....	13
<b>Instrucciones de uso</b> .....	13
<b>Capturas</b> .....	13
Análisis sintáctico .....	15
<b>Cambios en parser</b> .....	15
<b>Instrucciones de uso</b> .....	15
<b>Capturas</b> .....	16
Funciones auxiliares .....	17
Ejemplos .....	17
Conclusiones .....	18
Bibliografía .....	18

# Introducción

El presente trabajo tiene como objetivo realizar un analizador léxico y sintáctico del lenguaje “Pseudocódigo” usado en la Facultad Regional de Resistencia. En esta primera entrega se presenta la gramática a utilizar.

## Lenguaje de pseudocódigo

### Componentes léxicos

#### Palabras reservadas

```
# _div , _mod
# _o, _y, _no,
# leer,
# escribir
# si, entonces, sino, fin_si
# mientras, hacer, fin_mientras
# repetir, hasta_que
# para, hasta, fin_para
# segun, fin_segun
# accion, _es, fin_accion, proceso, ambiente
```

#### **Observaciones**

```
# No se distinguirá entre mayúsculas ni minúsculas.
# Las palabras reservadas no se podrán utilizar como identificadores.
```

#### Identificadores

##### **Características**

```
# Estarán compuestos por una serie de letras, dígitos y el guión bajo.
# Deben comenzar por una letra
# No podrán terminar con el símbolo de guión bajo, ni tener dos guión bajo seguidos.
# No se distinguirá entre mayúsculas ni minúsculas.
```

##### **Identificadores válidos:**

```
# dato, dato_1, dato_1_a
```

##### **Identificadores no válidos:**

```
# _dato, dato_, dato__1
```

## Tipos de Datos

- **Número**

# Se utilizarán números enteros, reales de punto fijo.

# Todos ellos serán tratados conjuntamente como números.

- **Cadena**

# Estará compuesta por una serie de caracteres delimitados por comillas dobles:

“Ejemplo de cadena”

“Ejemplo de cadena con salto de línea \n y tabulador \t”

# Deberá permitir la inclusión de la comilla doble utilizando la barra (\):

“Ejemplo de cadena con \" comillas\" simples”.

- **Operadores**

- **Operador de asignación**

# asignación: :=

- **Operadores aritméticos**

#suma: +

# Binario: 2 + 3

#resta: -

# Binario: 2 - 3

#producto: \*

#división real: /

#división entera: \_div

#módulo: \_mod

#potencia: \*\*

- **Operadores relacionales de números y cadenas:**

#menor que: <

#menor o igual que: <=

#mayor que: >

#mayor o igual: >=

#igual que: =

#distinto que: <>

**Por ejemplo:**

# si A es una variable numérica y control una variable alfanumérica, se pueden generar las siguientes expresiones relacionales:

(A >= 0) \_y (control <> “stop”)

- **Operadores lógicos**

#disyunción lógica: \_o

#conjunción lógica: \_y

#negación lógica: \_no

Por ejemplo: (A >= 0) \_y \_no (control <> “stop”)

- **Comentarios**

# De **encabezado** o descripción de file: delimitado por símbolos **/\*\* y \*/**  
*/\*\* Fantástico comentario de descripción de contenido de archivos  
puede incluir varias líneas \*/*

# De varias líneas: delimitados por los símbolos **/\* y \*/**  
*/\* ejemplo maravilloso  
de un comentario  
de tres líneas \*/*

# De una línea: Todo lo que siga al carácter **//** o **@** hasta el final de la línea.

// ejemplo espectacular de comentario de una línea

@ Otro ejemplo de línea

- **Fin de sentencia (implementación opcional)**

Punto y coma ;

# Se utilizará para indicar el fin de una sentencia.

## Sentencias

- **Inicio y fin de Algoritmo**

Accion identificador \_es  
sentencias

fin\_accion

# Declara nombre del algoritmo a utilizar, delimita las sentencias que forman el programa

- **Inicio de declaración de variables**

Ambiente

identificador : tipo de dato

# Declara nombre de variable de un tipo determinado

- **Inicio de declaración de sentencias**

Proceso

sentencias

# Declara inicio de proceso o conjunto de sentencias

- **Asignación**

identificador := expresión numérica

# Declara a identificador como una variable numérica y le asigna el valor de la expresión numérica.

# Las expresiones numéricas se formarán con números, variables numéricas y operadores numéricos.

identificador := “expresión alfanumérica”

# Declara a identificador como una variable alfanumérica y le asigna el valor de la expresión

alfanumérica.

- **Lectura**

Leer (identificador)

# Declara a identificador como variable numérica o alfanumérica y le asigna el caracter leído.

- **Escritura**

Escribir (expresión alfanumérica y/o identificadores)

# El valor de la expresión numérica es escrito en la pantalla.

# Se debe permitir la interpretación de comandos de saltos de línea (\n) y tabuladores (\t) que // tabs

puedan aparecer en la expresión alfanumérica.

escribir (“\t Introduzca el dato \n”);

escribir(“escribir texto mas identificador”, identificador)

- **Sentencias de control**

# Sentencia **condicional simple**

si (condición) entonces

sentencias

fin\_si

# Sentencia **condicional compuesta**

si (condición) entonces

sentencias

sino

sentencias

fin\_si

# Bucle **“mientras”**

mientras (condición) hacer

sentencias

fin\_mientras

#Una condición será una expresión relacional o una expresión lógica compuesta.

# Bucle **“repetir”**

repetir

sentencias

hasta\_que (condición)

# Bucle **“para”**

para (identificador:=valor\_inicial) hasta valor\_final, incremento hacer

sentencias

fin\_para

# Gramática

## Descripción de la gramática

AMBIENTE → *ambiente*

BLOQUE\_AMBIENTE → *conjunto de sentencias del ambiente*

BLOQ\_AMBIENTE → *línea dentro del ambiente*

IDENTIFICADOR → *identificadores o números*

VARIABLE → *variable*

CONSTANTE → *constante*

TIPO\_DATO → *tipo de dato*

PROCESO → *proceso*

CONJ\_SENTENCIA → *varias líneas de sentencias de pseudo*

SENTENCIA → *sentencia*

S\_ESCRIBIR → *sentencia escribir*

S\_SI → *sentencia si*

S\_CICLOS → *sentencia ciclos*

S\_SEGUN → *sentencia según*

SALIDA\_ESC → *salida de escribir*

ENTRADA\_LEER → *entrada de leer*

COMENTARIO\_ENCABEZADO → *comentario de tipo encabezado*

COMENTARIO\_VARIASLINEAS → *comentario de varias líneas*

COMENTARIO\_LINEA → *comentario de una línea*

C\_PARA → *ciclo para*

C\_MIENTRAS → *ciclo mientras*

C\_REPETIR → *ciclo repetir*

CONJ\_CONDICIONES → *conjunto de condiciones*

CONJ\_S\_SI → *conjunto de sentencias del SI*

CONJ\_COND\_SEGUN → *conjunto de condiciones SEGÚN*

ID\_TIPODATO → *identificador o tipo de dato*

CONJ\_OPERACIONES → *conjunto de operaciones*

OP\_ARITMETICA → *operación aritmética*

T\_OP\_ARITMETICO → *tipo de operador aritmético*

RELACIONALES → *relacionales*

T\_RELACIONAL → *tipo de relacional*

T\_OP\_LOGICO → *tipo de operadores lógicos*

OP\_CONDICION → *operación de condición*

EXPRESION → *expresión*

CONDICION → *condición*

## **Producciones**

**(!) IMPORTANTE: MAYUSCULA = TERMINALES; MINUSCULAS = NO TERMINALES.**

Aquellas derivaciones que se encuentran con color azul, es porque permiten la inclusión de comentarios.

$\Sigma \rightarrow$  ejecucion

ejecucion → COMENTARIO\_ENCABEZADO nombre FIN\_ACCION | COMENTARIO\_ENCABEZADO nombre bloque FIN\_ACCION comentario\_vl\_l | nombre bloque FIN\_ACCION comentario\_vl\_l | nombre bloque FIN\_ACCION

nombre → ACCION IDENTIFICADOR ES

comentario\_vl\_l → COMENTARIO\_VARIASLINEAS | COMENTARIO\_LINEA

BLOQUE → ambiente proceso

ambiente → AMBIENTE bloque\_ambiente | AMBIENTE comentario\_vl\_l bloque\_ambiente | AMBIENTE bloque\_ambiente comentario\_vl\_l

bloque\_ambiente → variable | constante | comentario\_vl\_l | Comentario\_vl\_l bloque\_ambiente | variable bloque\_ambiente | constante bloque\_ambiente

- ✓ Del bloque ambiente puede derivar en asignación de variables o constantes, hasta que no exista una nueva asignación.

variable → IDENTIFICADOR dos\_puntos ftd\_amb

ftd\_amb → TD\_ALFANUMERICO | TD\_NUMERICO | TD\_LOGICO

constante → IDENTIFICADOR IGUAL tipo\_dato

tipo\_dato → CADENA | NUMERICO

proceso → PROCESO conj\_sentencia

conj\_sentencia → s\_escribir | s\_leer | s\_si | s\_según | s\_ciclos | sentencia | comentario\_vl\_l | s\_escribir conj\_sentencia | s\_leer conj\_sentencia | s\_si conj\_sentencia | s\_según conj\_sentencia | s\_ciclos conj\_sentencia | sentencia conj\_sentencia | comentario\_vl\_l conj\_sentencia

- ✓ Dentro de CONJ\_SECUENCIA se derivan todas las acciones que se pueden realizar dentro del proceso del algoritmo.

s\_escribir → ESCRIBIR PARENTESIS\_ABIERTO salida\_esc PARENTESIS\_CERRADO

salida\_esc → IDENTIFICADOR | CADENA | IDENTIFICADOR COMA salida\_esc | CADENA COMA salida\_esc

- ✓ Desde S\_ESCRIBIR podemos obtener el “escribir” en pseudocódigo, mientras que SALIDA\_ESC contiene todo lo que se puede hacer en un escribir.



s\_leer → LEER PARENTESIS\_ABIERTO entrada\_leer PARENTESIS\_CERRADO  
entrada\_leer → IDENTIFICADOR

sentencia → IDENTIFICADOR ASIGNACION tipo\_dato | IDENTIFICADOR ASIGNACION  
op\_aritmetica  
id\_tipodato → IDENTIFICADOR | tipo\_dato

op\_aritmetica → id\_tipodato t\_op\_aritmetico id\_tipodato | id\_tipodato t\_op\_aritmetico

t\_op\_aritmetico → SUMA | RESTA | DIVISION | MULTIPLICACION | DIVISION\_ENTERA | MODULO |  
POTENCIA

t\_relacional → MENOR\_QUE | MAYOR\_QUE | MENOR\_O\_IGUAL\_QUE | MAYOR\_O\_IGUAL\_QUE |  
IGUAL | DISTINTO

t\_op\_logico → O | Y

- ✓ En estas producciones se definen todos los tipos de operadores para poder ser usados mas adelante.

s\_si → SI PARENTESIS\_ABIERTO conj\_condiciones PARENTESIS\_CERRADO ENTONCES conj\_s\_si  
FIN\_SI | SI PARENTESIS\_ABIERTO IDENTIFICADOR PARENTESIS\_CERRADO ENTONCES conj\_s\_si  
FIN\_SI

- ✓ En un condicional simple, se evalúa una condición o un conjunto de condiciones concatenado por operadores lógicos.

conj\_s\_si → conj\_sentencia | conj\_sentencia SINO conj\_si

- ✓ CONJ\_S\_SI esta producción tiene por finalidad dar la opción de anidar secuencias condicionales alternativas.

conj\_condiciones → condicion | condición t\_op\_logico conj\_condiciones

- ✓ De esta manera, se permite la comparación de verdad de una o varias condiciones con los operadores de disyunción y conjunción

condicion → expresion t\_relacional expresion | NO expresion

- ✓ Cada condición final es una operación que devuelve un booleano de verdadero o falso, los cuales son los operadores relacionales y el operador lógico de la negación

expresion → id\_tipodato | id\_tipodato t\_op\_aritmetico id\_tipodato  
| id\_tipodato t\_op\_aritmetico expresion

s\_según → SEGUN PARENTESIS ABIERTO IDENTIFICADOR PARENTESIS\_CERRADO HACER  
conj\_cond\_según FIN\_SEGUN

conj\_cond\_según → t\_relacional id\_tipodato DOS\_PUNTOS conj\_sentencia | t\_relacional  
id\_tipodato DOS\_PUNTOS conj\_sentencia conj\_cond\_según | OTRO DOS\_PUNTOS conj\_sentencia

s\_ciclo → c\_para | c\_mientras | c\_repetir

- ✓ Desde S\_CICLO se derivan todas las estructuras cíclicas que permite el pseudocódigo.

**c\_mientras** → MIENTRAS PARENTESIS\_ABIERTO conj\_condiciones PARENTESIS\_CERRADO HACER conj\_sentencia FIN\_MIENTRAS

**c\_repetir** → REPETIR conj\_sentencia HASTA\_QUE PARENTESIS\_ABIERTO conj\_condiciones PARENTESIS\_CERRADO

**c\_para** → PARA PARENTESIS\_ABIERTO idt\_para PARENTESIS\_CERRADO HASTA id\_tipodato COMA id\_tipodato COMA id\_tipodato HACER conj\_sentencia FIN\_PARA | PARA PARENTESIS\_ABIERTO idt\_para PARENTESIS\_CERRADO HASTA id\_tipodato HACER conj\_sentencia FIN\_PARA | PARA idt\_para HASTA id\_tipodato HACER conj\_sentencia FIN\_PARA

**idt\_para** → IDENTIFICADOR ASIGNACION id\_tipodato | IDENTIFICADOR

- ✓ Permitimos un tercer parámetro al PARA, donde puede variar el incremento en cada iteración.

## Terminales

- ACCION,
- ES,
- FIN\_ACCION,
- AMBIENTE,
- CADENA,
- NUMERICO,
- PROCESO,
- ESCRIBIR,
- LEER,
- O,
- Y,
- SI,
- ENTONCES,
- FIN\_SI,
- SINO,
- NO,
- HACER,
- SEGUN,
- FIN\_SEGUN,
- MIENTRAS,
- FIN\_MIENTRAS,
- OTRO,
- REPETIR,
- HASTA\_QUE,
- HASTA,
- PARA,
- FIN\_PARA,
- SUMA,
- RESTA,
- DIVISION,
- MULTIPLICACION,
- DIVISION\_ENTERA,
- MODULO,
- POTENCIA,
- IGUAL,
- MENOR\_QUE,
- MENOR\_O\_IGUAL\_QUE,
- MAYOR\_QUE,
- MAYOR\_O\_IGUAL\_QUE,
- DISTINTO,
- IDENTIFICADOR,
- DOS\_PUNTOS,
- ASIGNACION,
- COMA,
- SEMICOLON,
- COMENTARIO\_ENCABEZADO,
- COMENTARIO\_VARIASLINEAS,

- COMENTARIO\_LINEA,
- PARENTESIS\_ABIERTO,
- PARENTESIS\_CERRADO,
- TD\_NUMERICO,
- TD\_ALFANUMERICO,
- TD\_LOGICO

# Análisis léxico

Para la realización del LEXER acordamos el uso de la librería **PLY** para el lenguaje **Python**. Para entender esta librería usamos dos principales fuentes, la [documentación oficial](#) y un [ejemplo práctico](#) sencillo de cómo hacer una calculadora usando PLY, video que facilitó bastante el proceso, ya que nos ayudó a entender la estructura principal.

Lo primero fue transcribir todos los símbolos terminales o “**tokens**” dentro de un arreglo, y más tarde definir cada token con su correspondiente **expresión regular**. Algunas fueron definidas como **funciones** y otras como variables, debido a que las primeras tienen mayor “peso” u orden para el LEXER. Los nombres comienzan siempre con una “**t\_TERMINAL**” ya que la librería PLY identifica a los **tokens** de esta manera.

Para la creación de expresiones regulares nos apoyamos de la página [RegExr](#), la cual ayudó en comprobar que se cumplan los requisitos del lenguaje en cada token.

Realizamos **dos modos de ejecución**:

- uno línea por línea, el cual simplemente se inicia el programa y puede escribir en “modo libre”, mientras el lexer analiza cuando se presiona ENTER;
- y por último un modo de lectura de archivo de texto, el cual se acciona pasando como parametro “-f ruta\_archivo.e” por terminal. Esta opción genera un archivo txt llamado “tokens-analizados.txt”, el cual indica cada token y su valor, además de contar la cantidad total analizada y los errores léxicos.

## Instrucciones de uso

- Ir a la carpeta **bin**,
- abrir terminal y ejecutar programa con el comando “**lexer.exe**”.
- Para analizar un archivo de texto, pasar parámetros con “**lexer.exe -f ../prueba/HolaMundo.e**”.
- Para ejecutar el archivo **lexer.py** se debe instalar la librería “ply”, con el comando: **pip install ply**. Se pasan parámetros de la misma manera.

## Capturas

```
C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>lexer.exe
Pasa salir pulse: [ctrl] + [C] | O escriba _salir
>> escribir("prueba")
LexToken(ESCRIBIR, 'escribir', 1, 0)
LexToken(PARENTESIS_ABIERTO, '(', 1, 8)
LexToken(CADENA, 'prueba', 1, 9)
>> |
```

```
C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>lexer.exe -f ../prueba/HolaMundo.e
LexToken(COMENTARIO_ENCABEZADO,'/** Universidad Tecnologica Nacional Facultad Regional Resistencia Sintaxis y Semantic
a de los Lenguajes Ciclo:2021 Autor: Programa basico que imprime hola mundo */',1,0)
LexToken(ACCION,'ACCION',1,171)
LexToken(IDENTIFICADOR,'HolaMundo',1,178)
LexToken(ES,'_es',1,188)
LexToken(IDENTIFICADOR,'Ambiente',1,193)
LexToken(COMENTARIO_VARIASLINEAS,'/* Definimos el ambiente variables que usamos */',1,204)
```

```
C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>lexer.exe -f ../prueba/Errores.e
```

```
LexToken(ESCRIBIR,'escribir',1,0)
LexToken(PARENTESIS_ABIERTO,'(',1,8)
Caracter ilegal! : ' '.
En línea: 1. Posición: 9
LexToken(IDENTIFICADOR,'esta',1,10)
LexToken(IDENTIFICADOR,'cadena',1,15)
LexToken(IDENTIFICADOR,'no',1,22)
LexToken(IDENTIFICADOR,'estÃ',1,25)
Caracter ilegal! : '|'.
En línea: 1. Posición: 29
LexToken(IDENTIFICADOR,'cerrada',1,31)
Caracter ilegal! : '\'.
En línea: 1. Posición: 38
Caracter ilegal! : ' '.
En línea: 1. Posición: 39
LexToken(PARENTESIS_CERRADO,')',1,40)
(!) Se exportó un .txt con los tokens analizados.
```

```
C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>
```

tokens-analizados.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

TOKEN | VALOR

-----

```
1- ESCRIBIR: escribir
2- PARENTESIS_ABIERTO: (
3- IDENTIFICADOR: esta
4- IDENTIFICADOR: cadena
5- IDENTIFICADOR: no
6- IDENTIFICADOR: estÃ
7- IDENTIFICADOR: cerrada
8- PARENTESIS_CERRADO: )
-----
```

Total de tokens válidos analizados: 8.

Total de tokens NO válidos: 4.

# Análisis sintáctico

Para la realización del PARSER seguimos utilizando la librería **PLY** en **Python**. Ayudándonos de la documentación oficial, lo primero a realizar fue importar los tokens utilizados en el Lexer y luego comenzar a definir las producciones de la gramática, siguiendo tal cual se especifica en las páginas anteriores de este documento.

Para exportar los comentarios (en caso de haberlo), primero guardamos cada comentario en un array dentro del lexer y se lo pasamos al parser. De esta forma, si el parser analiza correctamente un archivo, recién allí exportará los comentarios obtenidos a un archivo .html, dentro de la carpeta 'pruebas'.

Cada vez que analiza un archivo, el parser generará un archivo de texto, donde se ve cada producción que se analiza. Esto nos ayuda entender qué llegó a analizar en caso de haber un error sintáctico en el archivo analizado.

## Cambios en parser

Invertimos el uso de mayúsculas, siendo ahora los *TERMINALES* o *TOKENS*, y las minúsculas ahora son las *producciones*, ya que esto evitaba confusiones con la terminología usada por el PARSER.

El token SEMICOLON (;) fue eliminado y se ignora en el lexer (dentro de `t_ignore`). Esto para que se pueda analizar un texto que contenga punto y coma y no presente errores en la gramática.

## Instrucciones de uso

- Ir a la carpeta **bin**,
- abrir terminal y ejecutar programa con el comando:
  - **"parserWindows.exe"**, si su sistema operativo es Windows.
  - **"chmod u+x parserLinux"**, luego **"./parserLinux"**, si está en Linux.
- Para analizar un archivo de texto, pasar parámetros con **"parserWindows.exe -f ../prueba/HolaMundo.e"**.
- En caso de no poder abrir el ejecutable, puede iniciar el archivo python (debe tener instalado Python en su computadora):
  - Ejecutar el archivo **parser.py** con **"python parser.py"**. No debe instalar la librería ya que está en la carpeta **"src/ply"**.
  - Se pasan parámetros de la misma manera que el ejecutable.

# Capturas

## Texto aceptado con HTML exportado

Universidad Tecnológica Nacional Facultad Regional Resistencia Sintaxis y Semantica de los Lenguajes Ciclo:2021 Autor: Programa basico que imprime hola mundo

Definimos el ambiente variables que usamos

clasico algoritmo de Hola Mundo

Imprimir por pantalla

Leer nombre usuario

Imprimir por pantalla

Imprimir por pantalla

Fin de programa

```
C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>
parserWindows.exe -f ..\pruebas\HolaMundo.e
Parser Pseudocodigo | Grupo 15. SSL 2021.
Ejecución completa!
(✓) Sintácticamente correcto. Se exportó un .html con los comen
tarios.
(!) Se exportó un .txt con las producciones analizadas.

C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>
```

```
File Edit Selection View Go Run ... parser-analisis.txt - ssl-pseudocodigo - Visual St...
bin parser-analisis.txt
28) Prod. Conjunto de sentencias --> | [conj_sentencia, s_escribir, conj_sentenci
29) Prod. Conjunto de sentencias --> | [conj_sentencia, comentario_vl_l, conj_sen
30) Prod. Conjunto de sentencias --> | [conj_sentencia, s_leer, conj_sentencia]
31) Prod. Conjunto de sentencias --> | [conj_sentencia, comentario_vl_l, conj_sen
32) Prod. Conjunto de sentencias --> | [conj_sentencia, s_escribir, conj_sentenci
33) Prod. proceso --> | [proceso, LexToken(PROCESO, 'Proceso', 1, 341), conj_senc
34) Prod. Bloque --> | [bloque, ambiente, proceso]
35) Prod. Comentario VL L --> | [comentario_vl_l, LexToken(COMENTARIO_LINEA, '// F
36) Prod. Ejecucion --> | [ejecucion, LexToken(COMENTARIO_ENCABEZADO, '/* Univers
37) Prod. Sigma --> | [sigma, ejecucion]
Total de tokens analizados: 37.
```

## Análisis con errores

```
bin > parser-analisis.txt
1 Producciones analizadas por el parser
2 -----
3 1) Prod. nombre accion --> | [nombre, LexToken(ACCION, 'ACCION', 1, 196), LexToken(IDENTIFICADOR, 'HolaMundo', 1, 203), LexToken(ES, 'es', 1, 213)]
4 -----
5 2) Prod. Comentario VL L --> | [comentario_vl_l, LexToken(COMENTARIO_VARIASLINEAS, '/* Definimos el ambiente variables que usamos */', 1, 229)]
6 -----
7 3) Prod. ftd_amb --> | [ftd_amb, LexToken(TD_ALFANUMERICO, 'cadena', 1, 298)]
8 -----
9 4) Prod. Variable --> | [variable, LexToken(IDENTIFICADOR, 'nombre_usuario', 1, 281), LexToken(DOS_PUNTOS, ':', 1, 296), ftd_amb]
10 -----
11 5) Error parser --> | LexToken(COMENTARIO_LINEA, '// variable sin definir tipo', 1, 319)
12 -----
13 Total de tokens analizados: 5.
14 -----
15
```

```
C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>parserWindows.ex
e -f ..\pruebas\HolaMundoError.e
Parser Pseudocodigo | Grupo 15. SSL 2021.
Error parser --> LexToken(COMENTARIO_LINEA, '// variable sin definir tipo', 1, 319
)
(!) Ocurrió un error sintáctico.
(!) Se exportó un .txt con las producciones analizadas.

C:\Users\juans\Documents\proyectos\python\ssl-pseudocodigo\bin>
```



# Funciones auxiliares

## En lexer:

Procedimiento **analizarTokens**(modoEjecucion: carácter):

- Analiza token por token.
- En caso de que el modo de ejecución sea “archivo” se guardará cada token en un arreglo para exportarlo a un txt posteriormente.

Procedimiento **exportarTokens**(arrAnalizar: arreglo):

- Del arreglo obtenido anteriormente, guarda cada elemento en un archivo txt, para facilitar el debugging.
- Además, imprime por pantalla si el archivo analizado es (o no) léxicamente correcto.

Arreglo “**arregloHtml**”:

- Almacenamos cada comentario que se encuentra en un archivo, indicando su tipo y valor. Ejemplo: [‘encabezado’, ‘/\*\*hola mundo\*/’].
- Este arreglo es importado luego en el parser.

## En parser:

Procedimiento **exportarHtml** (arregloHtml: arreglo):

- Se activa únicamente cuando analiza un archivo.
- Se importa “arregloHtml” y se realiza un recorrido del arreglo, verificando el tipo de comentario y así exportar a la etiqueta html correspondiente.
- El archivo html es creado en la misma ruta donde está el archivo de texto.

## Ejemplos

Se realizaron varios ejemplos tratando de utilizar todas las opciones que la gramática permite. Estos pueden ser encontrados en la carpeta “pruebas”.

# Conclusiones

Realizar el intérprete fue una tarea interesante y educativa, nos ayudó a entender cómo funcionan los distintos lenguajes de programación, y a crear nuevos. Además, se utilizó versionado con git y Github, lo cual facilitó la corrección de errores y el trabajo colaborativo. Creemos que este trabajo puede ser de gran utilidad para estudiantes de Algoritmos, ya que se podría continuarse haciendo un “compilador” y que ejecute código de verdad. De esta manera los estudiantes pueden verificar si sus algoritmos son correctos.

## Puntos fuertes

- Puede analizarse un archivo de texto desde cualquier ubicación, colocando la ruta correspondiente.
- Se realizaron pruebas con errores y el parser los rechaza correctamente.
- El lexer imprime el tipo de error, por ejemplo, si es un error de tipo IDENTIFICADOR o CADENA.
- Puede usarse punto y coma opcionalmente.

## Puntos débiles

- Los resultados del parser son un poco complicados de entender a simple vista.
- Para entender qué produjo un error semántico hay que conocer la gramática y ver las producciones analizadas anteriores al error.
- Hay advertencias de la librería ply acerca de dos producciones “ambiguas”, pero sin estas no funciona correctamente el análisis.

# Bibliografía

- [Stackoverflow](#),
- [Documentación PLY](#),
- [Regexr](#).