UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL RESISTENCIA

Trabajo Práctico Integrador: Intérprete de Pseudocódigo en Español

Grupo 15

Integrantes

Intile, Agustin

Salomón, Hilel Mauricio

Segnana, Juan Franco

Simonek, Joaquin

Zelinka, Gonzalo

Sintaxis y Semántica 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 3.1

Indice

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 sección 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 lineas */*

  # 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 linea

- 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→ *linea dentro del ambiente*
IDENTIFICADOR→ *identificadores o numeros*
VARIABLE→ *variable*
CONSTANTE→ *constante*
TIPO_DATO→ *tipo de dato*

PROCESO→ *proceso*
CONJ_SENTENCIA→ *varias lineas 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 lineas*
COMENTARIO_LINEA → *comentario de una linea*

C_PARA → *ciclo para*clea
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 → *operacion aritmética*
T_OP_ARITMETICO → *tipo de operador aritmético*

RELACIONALES → *relacionales*
T_RELACIONAL → *tipo de relacional*
T_OP_LOGICO → *tipo de operadores logicos*

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.
Σ → 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_segun | s_ciclos | sentencia | comentario_vl_l | s_escribir conj_sentencia | s_leer conj_sentencia | s_si conj_sentencia | s_segun 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 pseudocodigo, 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_segun → SEGUN PARENTESIS ABIERTO IDENTIFICADOR PARENTESIS_CERRADO HACER conj_cond_segun FIN_SEGUN

conj_cond_segun→ t_relacional id_tipodato DOS_PUNTOS conj_sentencia | t_relacional id_tipodato DOS_PUNTOS conj_sentencia conj_cond_segun | 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 pseudocodigo.

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 **t**okens de esta menara.

Para la creación de expresiones regulares nos apoyamos de la página RegExr, la cual ayudó en comprobar que se cumplan los requesitos 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 linea: 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 linea: 1. Posición: 29
LexToken(IDENTIFICADOR,'cerrada',1,31)
Caracter ilegal! : '\'.
En linea: 1. Posición: 38
Caracter ilegal! : '"'.
En linea: 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 Tecnologica 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

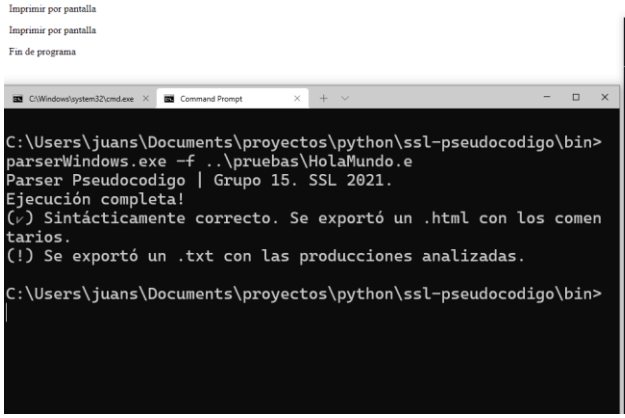
Imprimir por pantalla

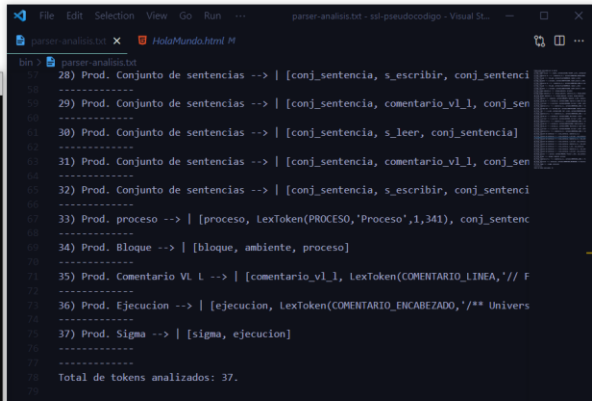
Leer nombre usuario

Imprimir por pantalla

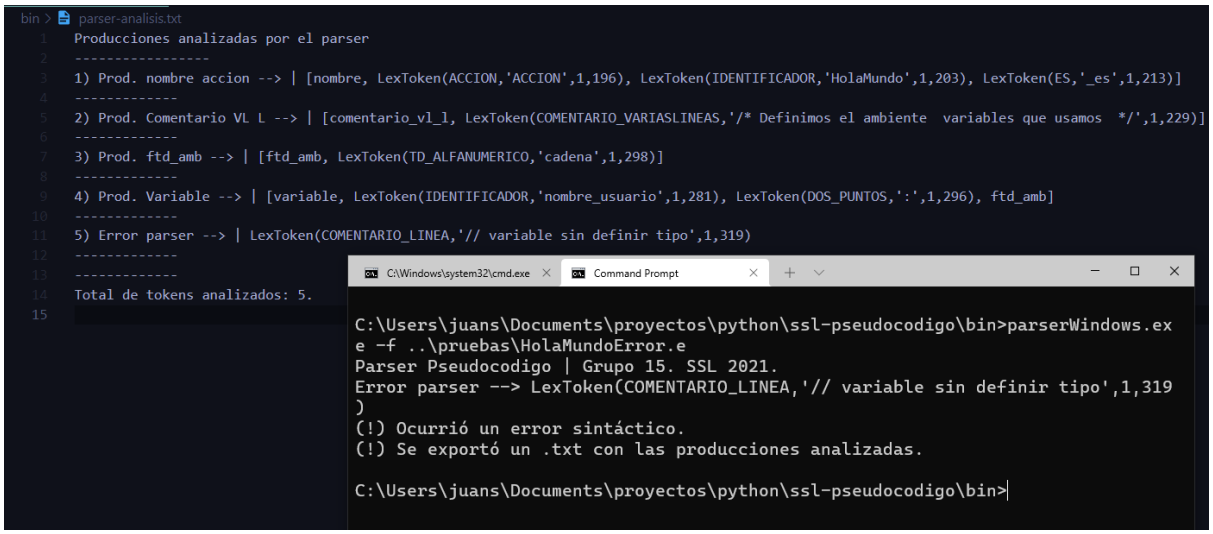
Imprimir por pantalla

Fin de programa





Análisis con errores



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". Además, se encuentran ejemplos propuestos por los profesores de la cátedra.

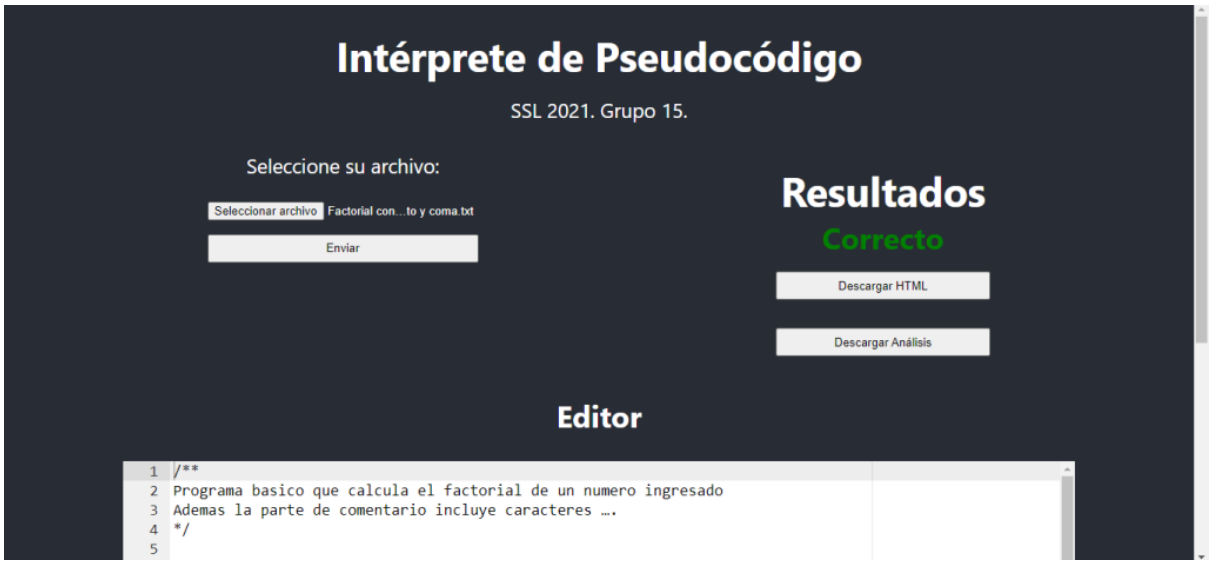
Página web

Para mejorar la carga/edición de archivos, se implementó una página web simple usando **ReactJS**, donde el usuario puede cargar un archivo de texto o bien usar un editor de texto, la página fue alojada en *Vercel*. Luego el texto se envía a un "servidor" creado con la librería **Flask** y alojado en *Heroku*, acá es donde se ejecuta nuestro lexer/parser. El servidor recibe el pseudocódigo y éste responde con:

- El texto es correcto/incorrecto,
- línea donde se encuentra el error,

- html resultante,
- análisis del parser, exportandose como un txt.

Página web:



Detección de error:

Link a página web. El código fuente de la página y el servidor estará disponible en este repositorio.

Instrucciones de uso localmente

- Ir a la carpeta **src/pagina/**. Debe tener instalado Nodejs. Abrir terminal e instalar dependencias de página web con comando: **npm install** .
- Para iniciar página web, ejecutar comando **npm start** . No usarla todavía.
- Ir a la carpeta **server**, instalar requerimentos con: **pip install -r requirements.txt** .

- Iniciar servidor con comando **python wsgi.py** . Esto creará un servidor local en dirección *http://localhost:5000/*, donde si se lo llama a la ruta **/pseudo** puede pasarse como argumento al pseudocódigo. Ejemplo: https://localhost:5000/pseudo?data=hola_mundo .
- Ahora puede entrar a la pagina (generalmente vía http://localhost:3000/) y puede comenzar a enviar pseudocódigo al servidor local corriendo el intérprete.
- En caso de que el servidor use otro puerto, puede modificarlo en el archivo **src/pagina/src/App.js**, línea 8.

Para evitar instalaciones localmente, alojamos tanto la página como al servidor en plataformas gratuitas. Vercel y Heroku, respectivamente. Los links de acceso son:

- Página: http://ssl-pseudopage.vercel.app/
- Servidor: http://ssl-pseudoserver.herokuapp.com/pseudo?data=

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.
- Edición de texto más rápida mediante la página web del intérprete.

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 producciones "ambiguas", siendo ambiguas algunas producciones que contienen comentarios. Tratando de eliminar la ambiguedad no funciona correctamente el análisis.

Bibliografía

- Stackoverflow,
- Documentación PLY,
- Regexr.
- Página web del intérprete.