VACACIONES DE DICIEMBRE 2021

MYCAREER-USAC

Manual Técnico

ELABORADO POR:

Rodrigo Alejandro Hernández de León
201900042

LENGUAJES FORMALES Y DE PROGRAMACIÓN - N

CONTENIDO

I. Introducción

El Programa de MyCareer-USAC es un programa de reconocimiento del lenguaje LFP realizando su respectivo análisis sintáctico y análisis léxico, realizado en el lenguaje de programación Python donde en este manual se explicará como se realizó el programa junto con las funciones utilizadas para que diera como resultado este pequeño analizador utilizando las bases del análisis léxico y sintáctico.

II. Objetivos

El objetivo primordial de este manual es ayudar a los distintos programadores y auxiliares de cátedra de los cursos de ciencias de la computación a informarse y guiarse de la comprensión del desarrollo de la aplicación hecha por el creador de este analizador de lenguaje LFP.

El objetivo de este sistema es poder comprender la utilidad del uso de gramáticas independientes de contexto y los árboles de derivación por medio de un análisis sintáctico junto con la comprensión de crear instrucciones de ejecución tipo de un lenguaje de programación.

III. Dirigido

Este manual esta orientado a todos los distintos programadores interesados en el desarrollo de esta aplicación y al auxiliar de cátedra del curso de lenguajes formales quienes tienen el conocimiento acerca de los temas de introducción a los compiladores.

IV. Especificación técnica

1. Requisitos de Hardware

- Computadora de escritorio o portátil.
- Mínimo 4GB de Memoria RAM.
- 250GB de Disco Duro o Superior.
- Procesador Inter Core i3 o superior.
- Resolución gráfica mínimo 1024 x 768 pixeles.

2. Requisitos de Software

- Sistema Operativo Windows 7 o superior
- Python en la versión 3.9.2
- Visual Studio Code u otro editor de texto.
- Librería externa PILLOW.
- Librería TKinter.
- Librería graphviz para la realización de grafos en Python.
- Navegador web para la lectura de archivos HTML.
- Lector de imágenes de formato .png.

V. Lógica del programa

Lenguaje LFP:

```
nombre_de_red('Nombre de la carrera');
crearcurso(No.semestre, codigo, 'Nombre del curso', []);
crearcurso(No.semestre, codigo, 'Nombre del curso', [Codigo]);
consola('texto');
consolaln('texto');
cursosPorSemestre(numero);
cursoPorCodigo(codigo);
cursoPorNombre('Nombre del curso');
cursosPrerrequisitos(codigo);
cursosPostrrequisitos(codigo);
generarRed('Nombre del archivo');
```

Análisis Léxico

1. Descripción del lenguaje LFP

Token	Descripción	Lexema
tk_nombre_de_red	Palabra reservada	nombre_de_red
tk_crearcurso	Palabra reservada	crearcurso
tk_consola	Palabra reservada	consola
tk_consolaln	Palabra reservada	consolaln
tk_cursosporsemestre	Palabra reservada	cursosPorSemestre
tk_cursoPorCodigo	Palabra reservada	cursoPorCodigo

tk_cursoPorNombre	Palabra reservada	cursoPorNombre
tk_cursosPrerrequisitos	Palabra reservada	cursosPrerrequisitos
tk_cursosPostrrequisitos	Palabra reservada	cursosPostrrequisitos
tk_generarRed	Palabra reservada	generarRed
tk_parentesisa	Carácter paréntesis que abre	(
tk_parentesisc	Carácter paréntesis que cierra	)
tk_puntoycoma	Carácter punto y coma	;
tk_coma	Carácter coma	,
tk_corchetea	Carácter corchete que abre	[
tk_corchetec	Carácter corchete que cierra	]
tk_cadena	Cadena de caracteres que empieza y termina con el carácter comilla simple	'esto es una cadena'
tk_entero	Secuencia de uno o más dígitos	101

2. Expresiones Regulares de los tokens

token	E.R.
tk_nombre_de_red	nombre_de_red
tk_crearcurso	crearcurso

tk_consola	consola
tk_consolaln	consolaln
tk_cursosporsemestre	cursosPorSemestre
tk_cursoPorCodigo	cursoPorCodigo
tk_cursoPorNombre	cursoPorNombre
tk_cursosPrerrequisitos	cursosPrerrequisitos
tk_cursosPostrrequisitos	cursosPostrrequisitos
tk_generarRed	generarRed
tk_parentesisa	\(
tk_parentesisc	\)
tk_puntoycoma	\;
tk_coma	\,
tk_corchetea	\[
tk_corchetec	\]
tk_cadena	\'.\'
tk_entero	[0-9]+

3. Método del Árbol

Expresión Regular:

1. P = Palabras Reservadas: [a-z][a-zA-Z_]

2. D = Dígitos= [0-9]

3. C = Caracteres= .

PP*#|\(#|DD*#|'C*'#|\,#|\[#|\]#|\)#|\;#

Árbol:

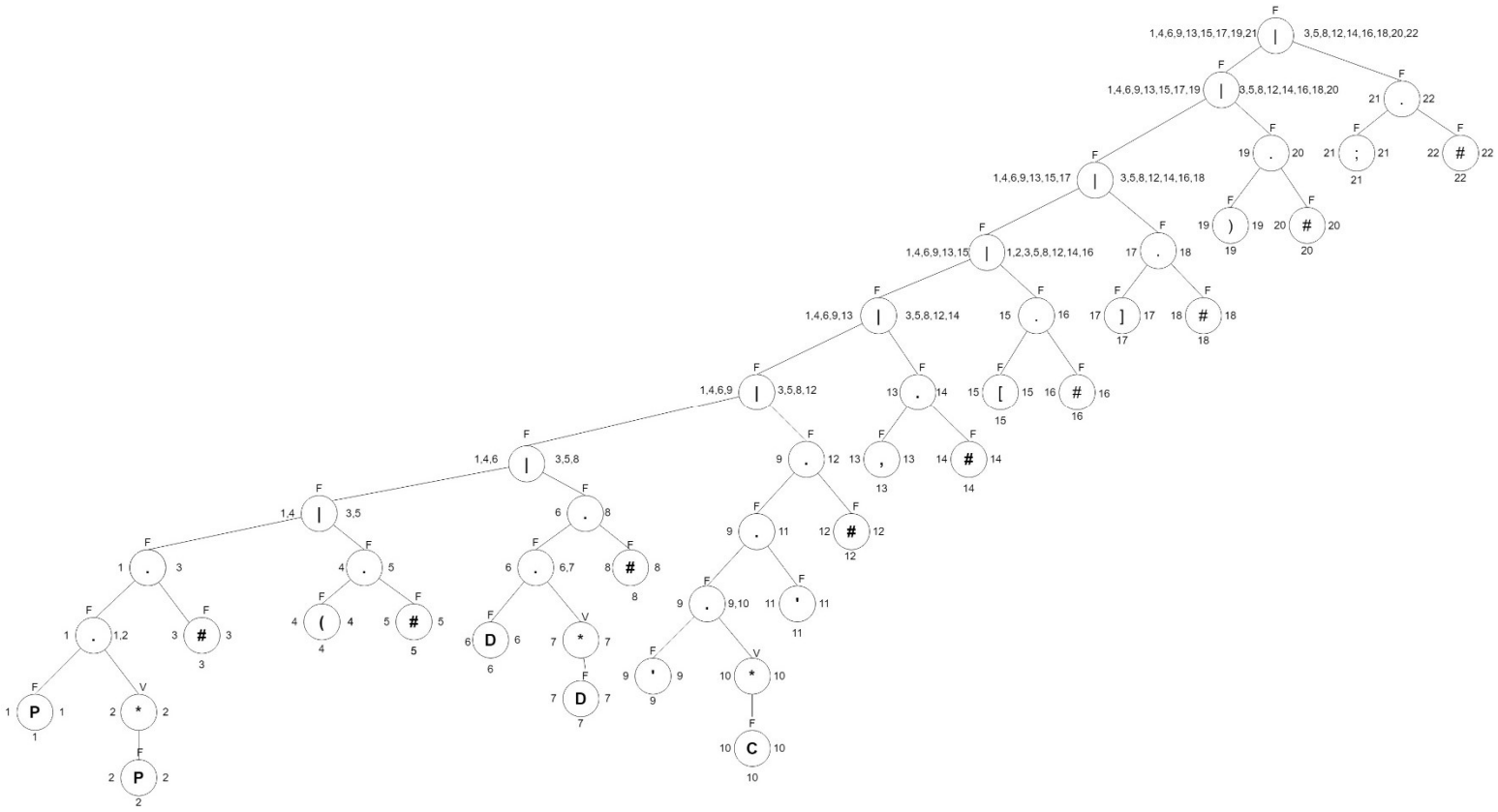
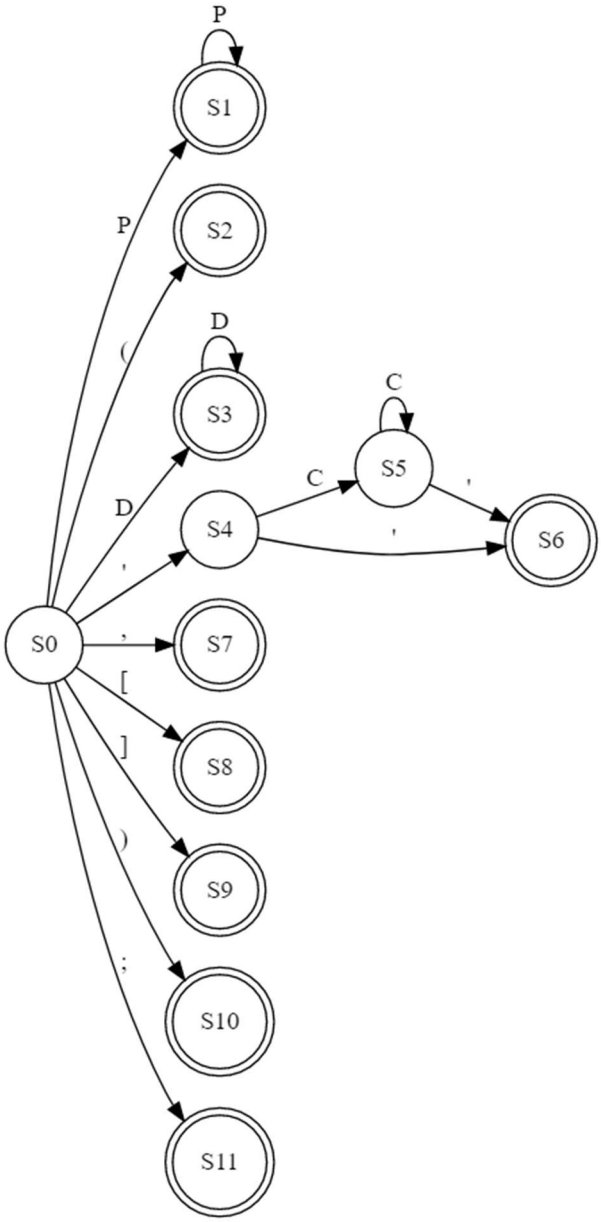


Tabla de siguientes:

Símbolo	Nodo	Siguiente
P	1	{2,3}
P	2	{2,3}
#	3	-
/(	4	{5}
#	5	-
D	6	{7,8}
D	7	{7,8}
#	8	-
\‘	9	{10,11}
C	10	{10,11}
\‘	11	{12}
#	12	-
\,	13	{14}
#	14	-

\[	15	{16}
#	16	-
\]	17	{18}
#	18	-
\)	19	{20}
#	20	-
\;	21	{22}
#	22	-

Autómata:



Análisis Sintáctico

Gramática Independiente del Contexto (Tipo 2):

```
                        GRAMATICA INDEPENDIENTE DEL CONTEXTO DEL LENGUAJE LFP
INICIO → INSTRUCCIONES

INSTRUCCIONES → INSTRUCCION INSTRUCCIONES2

INSTRUCCIONES2 → INSTRUCCION INSTRUCCIONES2
               | Epsilon << EOF >>

INSTRUCCION → NOMBRARRED
            | CREARCURSO
            | IMPRIMIRSINSALTO
            | IMPRIMIRCONSALTO
            | CURSOSPORSEMESTRE
            | CURSOPORCODIGO
            | CURSOPORNOMBRE
            | CURSOSPRERREQUISITOS
            | CURSOSPOSTRREQUISITOS
            | GENERARRED

NOMBRARRED → tk_nombre_de_red tk_parentesisa tk_cadena tk_parentesisc tk_puntoycoma
CREARCURSO → tk_crearcurso tk_parentesisa tk_entero tk_coma tk_entero tk_coma tk_cadena tk_coma
tk_ARREGLO tk_parentesisc tk_puntoycoma
IMPRIMIRSINSALTO → tk_consola tk_parentesisa tk_entero tk_parentesisc tk_puntoycoma
IMPRIMIRCONSALTO → tk_consolaln tk_parentesisa tk_entero tk_parentesisc tk_puntoycoma
CURSOSPORSEMESTRE → tk_cursosporsemestre tk_parentesisa tk_entero tk_parentesisc tk_puntoycoma
CURSOPORCODIGO → tk_cursoPorCodigo tk_parentesisa tk_entero tk_parentesisc tk_puntoycoma
CURSOPORNOMBRE → tk_cursoPorNombre tk_parentesisa tk_cadena tk_parentesisc tk_puntoycoma
CURSOSPRERREQUISITOS → tk_cursosPrerrequisitos tk_parentesisa tk_entero tk_parentesisc tk_puntoycoma
CURSOSPOSTRREQUISITOS → tk_cursosPostrrequisitos tk_parentesisa tk_entero tk_parentesisc
tk_puntoycoma
GENERARRED → tk_generarRed tk_parentesisa tk_cadena tk_parentesisc tk_puntoycoma

ARREGLO → tk_corchetea LISTAENTEROS tk_corchetec

LISTAENTEROS → tk_entero LISTAENTEROS2
             | Epsilon (tk_corchetec)

LISTAENTEROS2 → tk_coma tk_entero LISTAENTEROS2
              | Epsilon (tk_corchetec)
```

Código fuente:

1. Método AnalizarTexto()

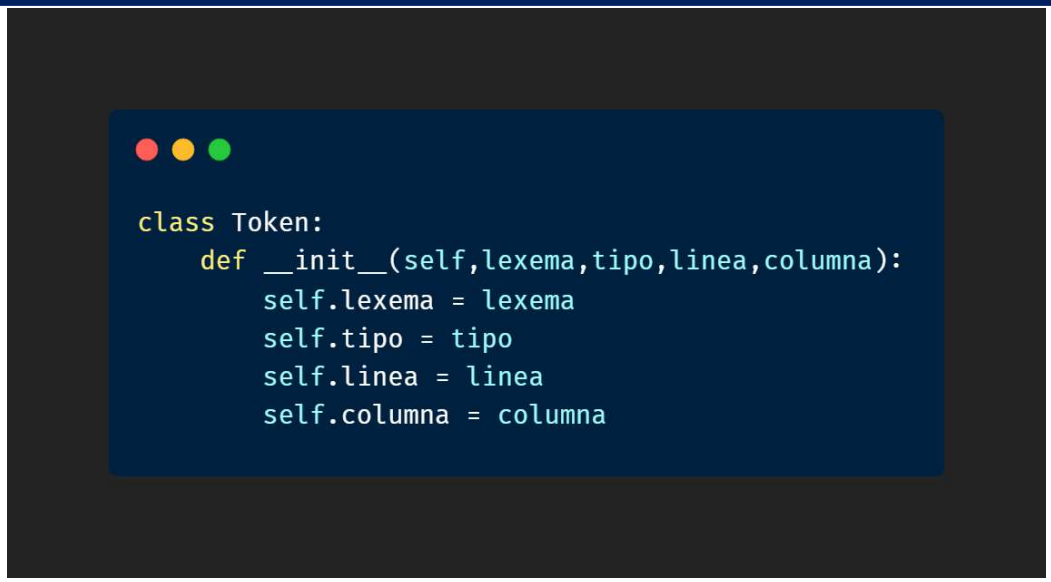
Se encuentra ubicado en el archivo app.py donde llama al analizador léxico y al analizador sintáctico conforme muestre los resultados de los análisis entonces realiza las distintas operaciones del lenguaje y genera los reportes correspondientes:

```
def AnalizarTexto():
    global cuadro1
    global escaner
    global cuadroconsola
    contenido = cuadro1.get(1.0, END)
    escaner.analisis(contenido)
    parser = AnalizadorSintactico()
    arbol = parser.analizar(escaner.listaTokens,escaner.listaErrores)
    arbol.ejecutar({})
    arbol.getNodos()
    if cuadroconsola.get(1.0, END) != "":
        cuadroconsola.config(state='normal')
        cuadroconsola.delete(1.0,END)
        cuadroconsola.insert(tk.INSERT, texto())
        cuadroconsola.config(state='disabled')
```

Para poder utilizar el analizador léxico fue útil el uso de estas dos clases:

2. Clase Token

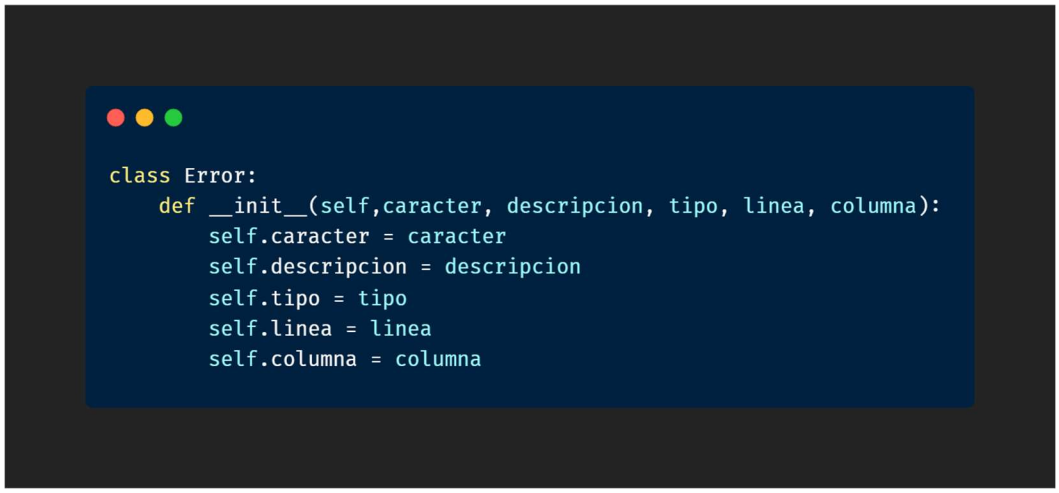
Esta clase hace que se guarden los objetos de tipo token aceptado.



```
class Token:
    def __init__(self,lexema,tipo,linea,columna):
        self.lexema = lexema
        self.tipo = tipo
        self.linea = linea
        self.columna = columna
```

3. Clase Error

Esta clase hace que se guarden los objetos donde haya un posible error léxico o sintáctico en su análisis.



```
class Error:
    def __init__(self,caracter, descripcion, tipo, linea, columna):
        self.caracter = caracter
        self.descripcion = descripcion
        self.tipo = tipo
        self.linea = linea
        self.columna = columna
```

Otra clase que fue útil pero en el desarrollo de la lógica del lenguaje que fue el uso de la clase Curso:

Clase Curso:

Esta clase se utiliza para guardar las propiedades del objeto curso que se crea al llamar la palabra reservada crearcurso.

```python
class Curso:
    def __init__(self, semestre, codigo, nombre, prerrequisitos):
        self.semestre = semestre
        self.codigo = codigo
        self.nombre = nombre
        self.prerrequisitos = prerrequisitos

    def getSemestre(self):
        return self.semestre

    def getCodigo(self):
        return self.codigo

    def getNombre(self):
        return self.nombre

    def getPrerrequisitos(self):
        return self.prerrequisitos
```

VI. Créditos

Elaborado por el alumno Rodrigo Alejandro Hernández de León para el curso de Lenguajes Formales y de Programación, en el país de Guatemala con fecha desde el 16 de diciembre de 2021 al 28 de diciembre de 2021.