Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Lenguajes Formales de Programación

Catedrático: Ing. Kevin Lajpop

Proyecto #2

Manual Técnico

Nombre: Javier Alexander Girón Donis **Registro Académico:** 202000510

Instructor: Bryan Gustavo Lopez Echeverría **Sección de Laboratorio:** N

Fecha de Realización: 30/12/2021 **Fecha de Entrega:** 30/12/2021

Objetivos y alcances del sistema

Objetivos generales:

1. Crear generador de mallas curriculares en él lenguaje de programación Python.
2. El manejo y comprensión de Tokens y Buffers.
3. Comprender y practicar los analizadores léxicos y sintácticos.

Objetivos específicos:

1. Elaborar un analizador léxico que cumpla con encontrar inconsistencias en él formato que se desea elaborar (.lfp).
2. Elaborar un analizador sintáctico que genere un orden especifico que deben llevar los tokens que se ingresen.
3. Comprender el funcionamiento de un analizador léxico para adquirir conocimientos sobre compiladores y sus componentes.
4. Poner en practica todos los conocimientos sobre el Método del árbol y así poder generar esa lógica de como ingresar una sucesión de caracteres que serán interpretados de manera correcta si se aplica de forma exitosa el concepto.

El generador de mallas curriculares tiene como finalidad poder ingresar cadenas de caracteres que serán analizadas de forma léxica y sintáctica para encontrar coherencia y ejecutar las instrucciones que tiene programadas si estas cumplen con estar de manera correcta.

Especificación Técnica

Requisitos de Hardware

- **Procesador:** Intel Core i5 (Intel Core i5 o equivalente)
- **RAM:** 2 GB, 4 GB (2 GB (32-bit), 4 GB (64-bit))
- **Disco duro:** 1.5 GB HDD
- **Tarjeta gráfica:** No necesario.
- **Resolución de la pantalla:** 1024 x 728

Requisitos de software

- **Sistemas Operativos:**
    - **Windows:** Windows 10, Windows 7, Windows XP, Windows Vista (Windows XP Professional SP3/Vista SP1/Windows 7 Professional).
    - **Mac:**
        - Mac con Intel que ejecuta Mac OS X 10.8.3+, 10.9+
        - Privilegios de administrador para la instalación
        - Explorador de 64 bits
    - **Linux:**
        - Oracle Linux 5.5+[1]
        - Oracle Linux 6.x (32 bits), 6.x (64 bits)[2]
        - Oracle Linux 7.x (64 bits)[2] (8u20 y superiores)
        - Red Hat Enterprise Linux 5.5+[1] 6.x (32 bits), 6.x (64 bits)[2]
        - Red Hat Enterprise Linux 7.x (64 bits)[2] (8u20 y superiores)
        - Suse Linux Enterprise Server 10 SP2+, 11.x
        - Suse Linux Enterprise Server 12.x (64 bits)[2] (8u31 y superiores)
        - Ubuntu Linux 12.04 LTS, 13.x
        - Ubuntu Linux 14.x (8u25 y superiores)
        - Ubuntu Linux 15.04 (8u45 y superiores)
        - Ubuntu Linux 15.10 (8u65 y superiores)
        - Exploradores: Firefox

- **Python 3:** En su versión más actual, se adjunta un enlace de descarga https://www.python.org/downloads/

- **Visual estudio Code:** Para poder revisar correctamente el código es necesario el uso de un IDE de programación el cual se recomienda Visual studio code al ser elaborado en este, se adjunta un enlace de descarga seguro Download Visual Studio Code - Mac, Linux, Windows

Almacenamiento de Tokens

Tokens.py: En este archivo podremos encontrar las clases para generar arrays bidimensionales donde se guardarán los tokens leídos, los errores al dar lectura al archivo ingresado y por ultimó se almacenarán las instrucciones encontradas en el archivo en 3 arrays. Los arrays son los siguientes:

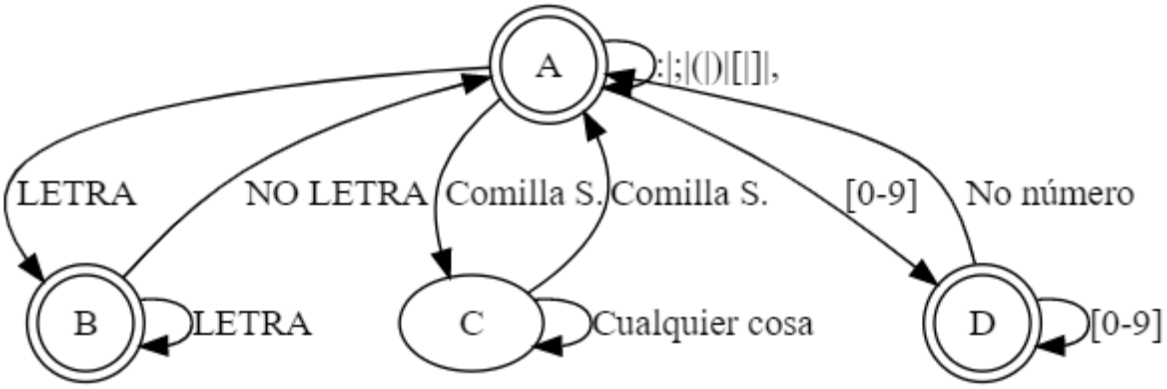
- **Token:** Almacenara todos los tokens obtenidos en el archivo.
- **Token_Error:** Almacenara los errores del análisis Léxico.
- **Token_Error_sintactico:** Almacenará los errores del análisis sintactico
- **opcionesD:** Guardara todos los cursos que sean ingresados
- **OpcionesB:** Almacenará todas las instrucciones generadas con una cadena de texto
- **OpcionesC:** Almacenará todas las instrucciones generadas con dígitos.

```
class Token_Error_sintactico(Token):
    def __init__(self, descripcion, tipo, fila, columna):
        self.descripcion = descripcion
        self.tipo = tipo
        self.fila = fila
        self.columna = columna

class opcionesD:
    def __init__(self, semestre, codigo, nombres,prerrequisitos):
        self.semestre = semestre
        self.codigo = codigo
        self.nombres = nombres
        self.prerrequisitos = prerrequisitos

class opcionesB:
    def __init__(self, instruccion, cadena):
        self.instruccion = instruccion
        self.cadena = cadena

class opcionesC:
    def __init__(self, instruccion, numeros):
        self.instruccion = instruccion
        self.numeros = numeros
```

Autómatas.py

Analizador Léxico: Tiene como funcionalidad crear estados desde la “A” hasta el estado “D” que cada uno cumplirá con un análisis en específico.

Para comprender mejor su funcionamiento tenemos el diagrama que sigue el autómata para cumplir el funcionamiento.



- **Estado A:** Su funcionalidad es encontrar todos los signos que no tengan que retornar ninguna funcionalidad.

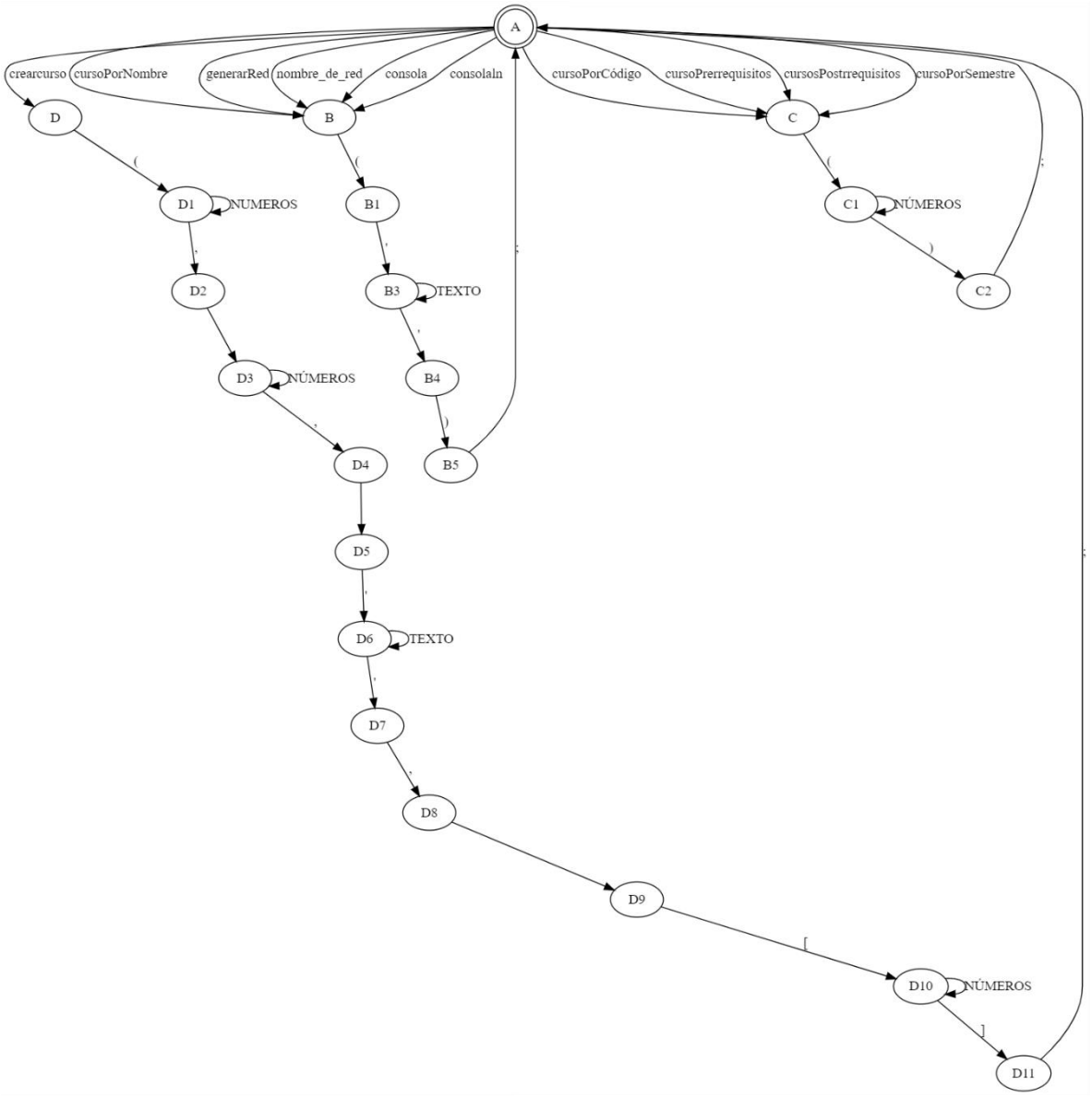
- **Estado B:** Cumple con la función de analizar las palabras reservadas que existen en el archivo y que se solicitan, estas son: nombre_de_red, crearcurso, consola, consolaln, cursoPorCodigo, cursoPorSemestre, cursoPorNombre, cursosPrerrequisitos, cursosPostrrequisitos y generarRed.

- **Estado C:** Su funcionalidad es poder almacenar toda cadena de texto contenida en comillas simples, cada palabra a partir de una de ellas será almacenada en el buffer para posteriormente ser guardada en el listado de tokens.

- **Estado D:** El ultimo estado cumple con la función de almacenar dígitos y que estos cumplan con una lógica.

Analizador Sintáctico: Dentro del mismo archivo .py podemos encontrar el analizador sintáctico, su función es recibir todos los tokens generados por el analizador léxico y organizarlos a modo de saber si estos están debidamente estructurados, la forma en la cual se creo este analizador fue siguiendo la misma lógica de un autómata teniendo estados que irán analizando los tokens y estos siguen el patrón correspondiente serán aceptados, caso contrario serán tomados como error.

Para comprender de mejor manera el autómata está el siguiente diagrama:



La parte principal que podemos notar a la hora de generar el análisis es que existen 3 etapas que son B, C y D. Su funcionamiento es el siguiente:

- **Estado B:** En el estado B podemos encontrar todas las instrucciones que lleven como ingreso una CADENA de texto, con ello deben seguir el patrón de ingreso donde esta su palabra reservada, paréntesis que abre, la cadena, cierre de paréntesis y punto y coma.
- **Estado C:** En el estado C podemos ingresar las instrucciones en las cuales es necesario almacenar un número, esta debe seguir el patrón de palabra reservada, paréntesis que abre, números ingresados, cierre de paréntesis y punto y coma.
- **Estado D:** Por último, tenemos el estado D que es el patrón mas largo ya que este almacenará todos los cursos que sean ingresados, en este el análisis es mas exigente ya que lleva tanto cadenas de texto como números así que es más extenso.