Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Lenguajes Formales de Programación

Catedrático: Ing. Kevin Lajpop

Proyecto #1

Manual Técnico

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

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

Fecha de Realización: 15/12/2021 **Fecha de Entrega:** 15/12/2021

Índice

Objetivos y alcances del sistema

Objetivos generales:

1. Crear un reproductor de música en él lenguaje de programación Python.
2. El manejo y comprensión de Tokens y Buffers.
3. Comprender y practicar los analizadores léxicos.

Objetivos específicos:

1. Elaborar un analizador léxico que cumpla con encontrar inconsistencias en él formato que se desea elaborar (mt3).
2. Comprender el funcionamiento de un analizador léxico para adquirir conocimientos sobre compiladores y sus componentes.
3. Poner en practica todos los conocimientos previamente adquiridos sobre compiladores tanto de la clase magistral como laboratorio.

El reproductor de música elaborado en el lenguaje de programación Python tiene como funcionalidad principal leer archivos no habituales como lo seria un nuevo formato como fue el caso del .mt3, cuenta con un analizador léxico que buscara errores o signos que no pertenezcan al formato establecido en los tokens.

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

Lógica del programa

Código Fuente:

- **Tokens.py:** En este archivo podremos encontrar dos clases las cuales hacen la creación de una lista bidimensional que servirá para almacenar tanto los tokens que se encuentran en él archivo analizado, como los tokens de errores encontrados en el mismo.

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

    def getinfo (self):
        print("Lexema: " + self.lexema + " Tipo: " + self.tipo + " Fila: " + str(self.fila) + " Columna: " + str(self.columna))

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

```

- **Automata.py:** En dicho archivo se encuentra sin duda la funcionalidad principal del programa, el cual es el “analizador léxico” que tiene como funcionalidad crear estados desde la “A” hasta el estado “E” que cada uno cumplirá con un análisis en específico:

    - **Estado A:** Su funcionalidad es encontrar todos los signos que no tengan que retornar ninguna funcionalidad, así mismo tiene algunos signos en especifico que al analizarlos hará que el estado cambie para iniciar una nueva acción.

```
26          if estado == 'A':
27
28              #Caracteres especiales
29              if caracter == '=':
30                  buffer += caracter
31                  columna += 1
32                  estado = 'A'
33                  self.listtoken.append(Token(buffer,'IGUAL', linea, columna))
34                  buffer = ""
35              elif caracter == '>':
36                  buffer += caracter
37                  columna += 1
38                  estado = 'A'
39                  self.listtoken.append(Token(buffer,'MAYOR', linea, columna))
40                  buffer = ""
41              elif caracter == ';':
42                  buffer += caracter
43                  columna += 1
44                  estado = 'A'
45                  self.listtoken.append(Token(buffer,'PUNTOYCOMA', linea, columna))
46                  buffer = ""
47              elif caracter == ':':
48                  buffer += caracter
49                  columna += 1
50                  estado = 'A'
51                  self.listtoken.append(Token(buffer,'DOSPUNTOS', linea, columna))
52                  buffer = ""
```

- **Estado B:** Cumple con la función de analizar las palabras reservadas que existen en el archivo y que se solicitan, estas son replist, nombre, artista, ruta, genero, repetir y año.

```
136         #Analisis de palabras reservadas
137         elif estado == 'B':
138             if caracter.isalpha() and not caracter.isnumeric():
139                 buffer += caracter
140                 columna += 1
141                 estado = 'B'
142             else:
143                 if buffer == 'REPLIST' or buffer == 'replist':
144                                 (variable) listtoken: list
145                     self.listtoken.append(Token(buffer,'REPLIST', linea, columna))
146                     estado = 'A'
147                     buffer = ""
148
149                 elif buffer == 'NOMBRE' or buffer == 'nombre':
150
151                     self.listtoken.append(Token(buffer,'NOMBRE', linea, columna))
152                     estado = 'A'
153                     buffer = ""
154
155
156                 elif buffer == 'ARTISTA' or buffer == 'artista':
157
158                     self.listtoken.append(Token(buffer,'ARTISTA', linea, columna))
159
160                     estado = 'A'
161                     buffer = ""
162
163                 elif buffer == 'RUTA' or buffer == 'ruta':
164
165                     self.listtoken.append(Token(buffer,'RUTA', linea, columna))
166
```

- **Estado C:** Parecido al estado B pero su funcionalidad es poder almacenar toda cadena de texto contenida en comillas dobles, cada palabra a partir de una de ellas será almacenada en el buffer para posteriormente ser guardada en el listado de tokens.

```
#Analisis de cadenas ""
elif estado == 'C':
    if caracter == '"':

        columna += 1
        estado = 'A'
        self.listtoken.append(Token(buffer,'CADENA', linea, columna))
        buffer = ""

    elif caracter != "'" and caracter != ',' and caracter != '\n':
        buffer += caracter
        columna += 1
        estado = 'C'

    else:
        self.listError.append(Token_Error('Error Sintáctico, no se cerro la cadena de texto', caracter, linea, columna))
        buffer = ""
        estado = 'A'
        indice += 1
```

- **Estado D:** Parecido al estado C pero 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.

```
#Analisis de cadenas ''
elif estado == 'D':
    if caracter == "'":
        columna += 1
        estado = 'A'
        self.listtoken.append(Token(buffer,'CADENA', linea, columna))
        buffer = ""


    elif caracter != "'" and caracter != ',' and caracter != '}' and caracter != '\n':
        buffer += caracter
        columna += 1
        estado = 'D'
    else:
        self.listError.append(Token_Error('Error Sintáctico, no se cerro la cadena de texto', caracter, linea, columna))
        buffer = ""
        estado = 'A'
        indice += 1
```

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

```
#Analisis de numeros
elif estado == 'E':
    if caracter.isdigit():
        buffer += caracter
        columna += 1
        estado = 'E'
    else:
        if caracter == ',':

            self.listtoken.append(Token(buffer,'NUMERO', linea, columna))
            self.listtoken.append(Token(",",'COMA', linea, columna))
            buffer = ""
            estado = 'A'
        else:
            self.listtoken.append(Token(buffer,'NUMERO', linea, columna))
            buffer = ""
            estado = 'A'
```

- **App.py:** Su función principal es la creación de la interfaz gráfica de todo el programa, todo esto se hace por medio de la librería llamada Tkinter y esta

por medio de sus funciones almacenadas como label, button, etc dará vida al reproductor.

Parte importante son sus clases y estas son:

- **Class UI(Frame):** Almacena todas las etiquetas y botones que contiene el reproductor.
- **leerArchivo:** Contiene la apertura del archivo que será utilizado para poder ser analizado posteriormente por el programa automata.py
- **abrir:** Con esta función podremos ejecutar lo que almacena el programa automata que una vez terminada la ejecución distribuirá los archivos por medio de arrays que serán utilizados para reproducir y mostrar las canciones con su respectiva información.
- **Iniciar:** Con la función iniciar daremos comienzo al reproductor y buscará las canciones almacenadas en el array guardar que tendrá cada una de las canciones encontradas en el archivo.