Proyecto 1

Analizador Léxico

Lenguaje C-

Descripción

Hacer un programa en Python, llamado **`lexer.py`** que contentan una función llamada **`getToken(imprime = true)`**, la cual, recibe una bandera booleana **`imprime`** , con valor por defecto **`true`** (su uso se explica más adelante) y regresa el siguiente token que encuentre en el string de entrada o un mensaje de error.

Utilice la implementación de variable **`estado`** o la de tabla. Recuerde que la implementación con variable **`estado`** es más eficiente.

Genere un archivo llamado **`globalTypes.py`**, por separado, que contenga todas los tipos de variables que se manejarán en su proyecto. Este archivo deberá estar en el mismo folder que el analizador léxico para que lo pueda importar sin problemas y sin necesidad de *paths* especiales.

El archivo globalTypes.py contendrá, entre otras cosas la definición del tipo enumerado:

`class TokenType(Enum):` la cual contendrá todos los tokens que se requieran (con sus valores) y en particular deberá contener el token **`ENDFILE`** (**`TokenType.ENDFILE`**), para que se puede probar sin problemas (ver "prueba del programa").

Desde luego que podrá contener todos los tipos enumerados que se requieran para el buen funcionamiento del analizador léxico.

El programa **`lexer.py`**, al inicio, deberá importar el archivo que contenga los tipos a ser manejados, de la siguiente forma:

```
from globalTypes import *
```

Posteriormente, continuará con la definición de la función **`getToken(imprime = true)`** y de todas las funciones auxiliares que requiera.

La función **`getToken(imprime = true)`**:

- Deberá declarar algunas variables globales, (ver la sección de "prueba del programa").
- Mediante esas variables podrá manejar el archivo de texto conteniendo un programa en C- que se desea analizar.
- Cada vez que sea invocada regresará el siguiente token que encuentre es decir, el par (TOKEN, Lexema), o un error, cuando este sea el caso (ver "Detección y recuperación de errores"). Su invocación se hará de la siguiente forma:

  ```
  token, tokenString = getToken(true)
  ```

  Por lo que la función regresará los dos valores por separado, de la siguiente forma:

  ```
  return token, tokenString
  ```

- Desde luego, **`token`** es de tipo **`TokenType`** y **`tokenString`** es de tipo **`String`** y corresponde al **`token.val`**, pero al recibirlo así evita tener que obtener su valor.
- Si la bandera **`imprime`** es true, la función deberá imprimir la pareja (TOKEN, lexema) encontrada, antes de regresarla. Una opción sería hacerlo así:

```
print(token," = ", tokenString)
```

Detección y recuperación de errores

Si el analizador léxico encuentra un error (recuerde que el analizador léxico sólo encuentra errores en la formación de los tokens, no dice nada con respecto a la sintaxis del lenguaje), debe marcar la línea y el lugar (caracter) donde encontró el error. Por ejemplo, si la línea 22 del archivo es:

```
contador = contador + 3indice
```

detectará los tokens:

```
(ID, "contador")
(EQUAL, "=")
(ID, "contador")
(PLUS, "+")
(ERROR, "")
```

Mandando un mensaje de error como:

```
Línea 22: Error en la formación de un entero:
contador = contador + 3indice
                      ^
```

Donde se indica el número de línea y el acento circunflejo (^) indicará la posición del carácter donde encontró el error.

Después de esto, deberá tener un mecanismo para tratar de recuperarse del error y continuar con la detección de tokens. Desde luego que, nada garantiza la exactitud de lo que se detecte después, lo cual dependerá del mecanismo utilizado y, sobre todo, de la intención que tenga el programador al escribir el código, la cual desconocemos por completo y sólo la podemos suponer.

Prueba del programa

Todos los archivos necesarios para que corra (todos lo que se entregan en Bb y el que contiene el archivo de prueba) se colocarán en la misma carpeta para evitar el uso de paths adicionales.

El analizador léxico será probado con un script que iniciará importando tanto el archivo con los tipos globales como el que contiene su analizador de léxico. Posteriormente seguirá invocando a su función para obtener un token, con la opción de impresión activada, hasta que se termine el archivo, es decir, hasta que llegue el token que indica el fin de archivo.

El script con el que se prueba será el siguente:

```
from globalTypes import *
from lexer import *

f = open('sample.c-', 'r')
programa = f.read()          # lee todo el archivo a compilar
progLong = len(programa)     # longitud original del programa
programa = programa + '$'    # agregar un caracter $ que represente EOF
posicion = 0                 # posición del caracter actual del string

token, tokenString = getToken(true)
while (token != TokenType.ENDFILE):
    tok, tokenString = getToken(true)
```

Por lo que la función **getToken(imprime)** deberá manejar las siguientes variables globales:

posicion: contiene la posición del siguiente carácter del string que se debe analizar. Deberá poder modificarla en su funcionamiento.

progLong: contiene la logitud del programa. Sólo la leerá cuando la requiera.

programa: contiene el string del programa completo. Sólo la leerá cuando lo requiera.

Para la entrega

- Un documento con:
  - Las expresiones regulares para detectar todos los tokens.
  - El DFA implementado que realiza la detección de tokens.
- Todos los archivo Python (comentados) y TXT necesarios para que corra.

Para crear el DFA no se requiere hacer la conversión de expresiones regulares a DFA. Se puede crear a mano, siempre y cuando genere los mismos tokens que las expresiones regulares.

El manual del usuario no será necesario porque ya se dio la definición del lenguaje C- y la forma en la que se probará el programa.

Si este proyecto se fuera a entregar a un tercero, es indispensable entregarla toda la definición del lenguaje y la forma en la que el usuario deberá usar el analizador de léxico, paso a paso.

Nota FINAL:

Este analizador léxico también puede generarse utilizando una herramienta de generación automática como Lex o Flex. Sin embargo, tendría que hacer las adecuaciones (recuerde que estos generadores dan como salida código en C) para que funcione tal como se realizó la descripción anterior de “prueba del programa”.