COMPILADORES
LABORATORIO 1

Laboratorio de Análisis Léxico- Lex

Generalidades

Objetivo: Aplicar el proceso de análisis léxico para identificar los tokens de un subconjunto del lenguaje Python y los errores léxicos.

Herramienta: Lex o Flex y C bajo Linux Ubuntu.
Entrada: Archivo de texto con un programa en Python

Salida: Archivo de texto con el programa en Python donde se identifique cada token y se listen los identificadores y el número de error encontrados.
Manual: Se debe incluir un manual de instrucciones.
Fecha de inicio: Febrero 14 de 2.020
Fecha de entrega: Marzo 16 de 2.020
Nombre programa: LAB01_Apellido1_Apellido2 (Fuente Lex).
LAB01_Apellido1_Apellido2 (Fuente C).
LAB01_Apellido1_Apellido2 (Ejecutable)
Los archivos se deben entregar comprimidos en un archivo llamado **LAB01_Apellido1_Apellido2.zip. No se acepta en otro formato de compresión.** El manual debe ser incluido en este archivo comprimido.

Integrantes: 3 ESTUDIANTES POR GRUPO (Sustentan todos los integrantes del grupo).

Descripción de la gramática

El analizador léxico recibirá un archivo de texto con el programa en Python que puede contener los siguientes tokens:

- ✓ *Palabras claves o reservadas:*

```
and             else            is              return
break           for             not             while
continue        if              or
def             import          pass
elif            in              print
```

- ✓ *Operadores:*

```
+       -       *       **      /       //      %
<<      >>      &       |       ^       ~
<       >       <=      >=      ==      !=      <>
```

✓ *Identificadores:* Comienzan con letra o '_' y siguen cero o más letras, dígitos o '_'.

✓ *Números:*

- Enteros:
  - integer: + o - uno o más dígitos
  - longinteger: integer(L | l)
- Decimales: ej: 3.14 10. .001 1e100 3.14e-10 0e0
- Imaginario: comienza con un número entero o decimal, seguido de una 'j' o 'J'.

✓ *Strings:* comienzan con ' o " y contienen cualquier caracter excepto ' , " o \n.

✓ *Delimitadores*

```
(       )       [       ]       =       ;
,       :       .       >>=     <<=
+=      -=      *=      /=      //=
&=      |=      ^=      **=     %=
```

✓ *Comentarios:* comienzan con # y terminan al final de la línea. Los comentarios son ignorados.

Estructuras

Se especifica a continuación la estructura de las instrucciones que se evaluarán, ignorando muchas de las funciones que maneja Python. Lo que se encuentra en [*CURSIVA*] implica un componente opcional, lo que se encuentra en **negrilla** implica una palabra reservada. Un '**[**' o '**]**' en negrita indica que estos caracteres van en la estructura, y no son opcionales.

Asignación

Identificador = expresión
Identificador1 *[, identificador2, …]*= expresión1 *[, expresión1, …]*

Donde una expresión es una combinación a través de operadores de las siguientes posibilidades:

- Expresión aritmética
- Expresión booleana: usa **True**, **False**, identificadores, operadores de comparación y booleanos como **is**, **not** , **and** , >= , <, etc
- String constante
- Identificadores

- Lista: **[** *[expresión1, expresión2,…]* **]**
- Posición en una lista: identificador**[**expresión_artimética**]**
- Llamada de funciones: identificador((*[expresión1, expresión2, expresión3,…]*))

Definición de función

def identificador ([*paramétro1, parámetro2, …*]):
sentencia(s)
*[**return** expresión]*

Donde los parámetros son identificadores.

Condicional

if expresión_booleana :
sentencia(s)
[*elif* *expresión booleana :*
sentencia(s)
elif *expresión booleana :*
sentencia(s)

.
.
.
else*:*
sentencia(s)]

Ciclo for

for identificador **in** secuencia:
sentencia(s)

Ciclo while

while expresión_booleana:
sentencia(s)

Donde una secuencia es una lista o un **range**

Pass

pass

Break

break

Contn

continue

Print

print(*[expresión1, expresión2, expresión3,…]*)

Import

import identificador

Range

range(expresión*[, expresión]*)

Descripción del archivo de salida

La salida debe mostrar las instrucciones del archivo de entrada identificando el tipo de token encontrado a excepción de las palabras reservadas. A continuación, se detallan las especificaciones:

- ✓ Cada una de las palabras reservadas debe mostrarse en mayúscula
- ✓ Los identificadores deben ir enumerados. En caso de encontrar uno repetido debe tener la misma numeración en cada aparición. Al terminar el análisis léxico se deben mostrar cuántos y cuáles fueron los identificadores encontrados.
- ✓ Para los números, operadores y separadores se debe especificar el tipo de token encontrado.
- ✓ Para los números identificar si son enteros, decimales o imaginarios

Errores Léxicos

- ✓ Un token no especificado en la gramática debe indicar error en el lugar donde fue encontrado.
- ✓ Un error en la estructura de la instrucción no es un error léxico
- ✓ Si hay error léxico en algún punto del archivo, se debe continuar el análisis hasta el final.
- ✓ Al terminar el análisis se debe mostrar cuántos errores fueron encontrados.

Ejemplos

Entrada

```
def eval(xi, exp):
        ans = False
        for i in range(len(exp)) :
                part =? True
                for j in range(len(exp[i])) :
                        part = part and xi[exp[i][j]]
                ans = ans or part

        return ans
# comentario ignorado

while 2*i<=40000L
        h = 3.4e-5 + 5j
```

```
        if i == 0:
                break
def otra_fun():
        pass
print('Esta es la respuesta',con, 'y la expresión ', 3*i+5*(7-x))

#string de la hipotesis

table = []
y = []
#i = 0
```

Salida

```
DEF id1=eval parabre=(id2=xi coma=, id3=exp parcierr=) dospunt=:
        id4=ans asig= = FALSE
        FOR id5=i in RANGE parabre=( id6=len parabre=( id3=exp parcierr=) parcierr=) dospunt=:
                Id7=part ERROR= =? TRUE
                FOR id8=j IN RANGE parabre=( id6=len parabre=( id3=exp corabre=[ id5=i corcierr=]
parcierr=) parcierr=) dospunt=:
                        Id7=part assign= = id7= part AND id2=xi corabre=[ id3=exp corabre=[ id5=i
corcierr=]  corabre=[ id8=j corcierr=] corcierr=]
                Id4=ans asign= = id4=ans OR id7=part

        RETURN id4=ans

WHILE entero=2 mult=* id5=I menor_ig = <= long=40000L
        Id9=h assign= = real= 3.4e-5 suma= + imaginario=5j
        IF id5=i comp= == entero=0 dospunt= :
                BREAK
DEF id10= otra_fun  parabre=( parcierr=) dospunt=:
        PASS

PRINT parabre=( cadena='Esta es la respuesta' coma=, id11=con_123 coma=, cadena='y la expresión '
coma=, entero=3 mult= * id5=i suma=+ entero=5 mult=* parabre=( entero=7 menos= - id12=x
parcierr=) parcierr=)

Id13=table asign= = corabre=[ corcierr=]
Id14=y asign= = ERROR= 123ABC
```

14 Identificadores

Id1=eval
Id2=xi
Id3=exp
Id4=ans
Id5=i

Id6=len
Id7=part
Id8=j
Id9=h
Id10=otra_fun
Id11=con_123
Id12=x
Id13=table
Id14=y

2 Errores