COMPILADORES
LABORATORIO 2

Laboratorio de Análisis Sintáctico- Yacc

Generalidades

Objetivo: Aplicar el proceso de análisis sintáctico para identificar la veracidad sintáctica de un subconjunto del lenguaje Python y determinación de los errores sintácticos.

Integrantes: 4 ESTUDIANTES POR GRUPO.
Herramienta: Lex, Yacc y C bajo Linux.
Entrada: Un archivo llamado **entrada.txt**, el cual contenga un programa con las indicaciones del laboratorio anterior y basado en el lenguaje de programación Python propuesto.
Salida: Generación de un archivo de salida llamado **salida.txt** a continuación del análisis sintáctico. Esta salida contendrá el número de cada línea errada del archivo de prueba o dirá si el programa está bien escrito sintácticamente.

Manual: Se debe incluir un manual de instrucciones de ejecución paso a paso.
Fecha de inicio: Octubre 14 de 2024
Fecha de entrega: Noviembre 8 de 2024
Nombre programa: LAB03_Apellido1_Apellido2_Apellido3_Apellido4 (Fuente Lex).
LAB03_Apellido1_Apellido2_Apellido3_Apellido4 (Fuente Yacc).
LAB03_Apellido1_Apellido2_Apellido3_Apellido4 (Ejecutable).

Los archivos se deben entregar comprimidos en un archivo llamado **LAB03_Apellido1_Apellido2.zip. No se acepta en otro formato de compresión.** El manual debe ser incluido en este archivo comprimido.

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 ([*parámetro1, 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]*)

Análisis Léxico

Contemplar las mismas especificaciones del laboratorio anterior.

Análisis Sintáctico

Entrada: El mismo archivo de prueba anterior con cualquier extensión, el cual contiene un programa con las indicaciones anteriores o con errores sintácticos que se deben detectar.

Salida: Generación de un archivo de salida a continuación del análisis sintáctico. Esta salida contendrá la o las líneas erradas del archivo de prueba o dirá si el programa está bien escrito sintácticamente.

Ejemplos

1. Dado el siguiente programa en el archivo de entrada **prueba1.py**

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
```

La salida en **saliday.txt** es:
Prueba con el archivo de entrada
0 errores

2. Dado el siguiente programa en el archivo de entrada **prueba2.py**

Entrada

```
def eval(xi, exp)
        ans = False
        a,b = 3
        for i in range len(exp)) :
                part = True
                for j range(len(exp[i])) :
                        part = part and xi[exp[i][j]]
                ans = ans or part
```

```
        return ans
```

La salida en **saliday.txt** es:

```
Prueba con el archivo de entrada
línea 1 error
línea 3 error
línea 4 error
```