**UTN-FRD**

**Trabajo Practico Nro. 1**

**Sintaxis y semántica de los lenguajes**

**Alumno:** Julian Vazquez

**Docente:** Hernán Miranda.

**Objetivo**: Implementar un analizador lexicográfico para una gramática especificada.

**La gramática G=<VN,T,P,S>:**

VT = {id, num, si, entonces, sino, finsi, repetir, hasta, equal, leer, mostrar, func, finfunc, (, ), ;, oprel, opsuma, opmult}

VN = {Program, ListaSentencias, Sentencia, SentenciaSi, SentenciaRepetir, SentenciaAsig, SentenciaLeer, SentenciaMostrar, SentenciaFun,Proc, ListaPar, Expression, Expresion2, Factor, Termino}

S = Program

P = {

Program → ListaSentencias

ListaSentencias → ListaSentencias;Sentencia | Sentencia

Sentencia → SentenciaSi | SentenciaRepetir | SentenciaAsig | SentenciaLeer | SentenciaMostrar | SentenciaFun

SentenciaSi → “si” Expresion “entonces” ListaSentencias “sino” ListaSentencias “finsi” | “si” Expresion “entonces” ListaSentencias “finsi”

SentenciaRepetir → “repetir” ListaSentencias “hasta” Expresion

SentenciaAsig → “id” “equal” Expresion

SentenciaLeer → “leer” “id”

SentenciaAsig → “mostrar” Expresion

SentenciaFun → “func” Proc “finfunc”

Proc → “id” “(“ ListaPar “)” ListaSentencias

ListaPar → ListaPar “;” “id” | “id”

Expresion → Expresion2 “oprel” Expresion2 | Expresion2

Expresion2 → Expresion2 “opsuma” Termino | Termino

Termino → Termino “opmult” Factor | Factor

Factor → “(“ Expresion “)” | “num” | “id”

}

**Explicación del algoritmo:**

El algoritmo comienza con un diccionario que define Automata Finito Deterministico o “DFA”, estos automatas obvian el estado trampa y ante cualquier transición no definida, esta va a el estado trampa. Con estos automatas definidos, pasamos al lexer definido como “lexer(programa)” program siendo un string. A este string se le eliminan los espacios en blanco al principio y al final del string, acto seguido se le agrega un espacio para facilitar la busqueda del ultimo token. Se inician tres listas “tokens\_out” que almacena los tokens lo de salida, “tokens\_posibles” que contiene los tokens posibles para el lexema actual y “tokens\_posibles\_1mas” que es una copia de “tokens\_posibles” y se usa para analizar el siguiente carácter. Se inician dos strings, “lexema” que almacena el lexema actual y “lexema1mas” que se usa para construir el lexema del siguiente carácter. Se recorre todos los caracteres del string “programa”, se agrega a “lexema” el contenido de “lexema1mas” y a “lexema1mas” se le agrega el carácter actual de “programa”. En este punto se inicia una bifurcación, donde si la lista “tokens\_posibles” esta vacia, significa que se acaba de agregar un lexema a la salida, por lo que se necesita recalcular la lista “tokens\_posibles”, esto se hace comparando el proximo lexema con los “DFA” de cada token para determinar el lexema es valido con ese token. Si “tokens\_posibles” no esta vacia, se copia el valor de “tokens\_posibles\_1mas” del ciclo anterior. Se verifica si “lexema” es un espacio en blanco, salto de línea o tabulado, si eso se cumple se saltea el carácter y se continua con el siguiente. Se forma una lista llamada “tokens\_final” que almacena los tokens que llegan a un estado final, en su automata asociado, que renoconen el lexema actual. Para cada token en “tokens\_posibles”, se utiliza su “DFA” para determinar si el estado actual del lexema y el estado con un carácter mas. Si el estado con un carácter mas es un estado trampa, se elimina ese token de “tokens\_posibles\_1mas”. Si el estado actual es final el token a la lista “tokens\_final”. Al final si “tokens\_posibles\_1mas” está vacia, si “tokens\_final” no es una lista vacia, se agrega el primer elemento de esta ultima a la lista “tokens\_out” junto con su lexema correspondiente, por ultimo caso si “tokens\_final” esta vacia quiere decir que no existe automata que reconozca ese token, en ese caso se retorna el mensaje “Token invalido”. Este proceso se repite hasta terminar con el string programa.

**Ejemplos o pruebas realizadas:**

Prueba 1:

lexer("repetir x equal 5 finsi 123x x>=3 fin finfunc func")

[('repetir', 'repetir'),

('id', 'x'),

('equal', 'equal'),

('numero', '5'),

('finsi', 'finsi'),

('numero', '123'),

('id', 'x'),

('id', 'x'),

('oprel', '>='),

('numero', '3'),

('id', 'fin'),

('finfunc', 'finfunc'),

('func', 'func')]

Prueba 2:

lexer("i equal 2")

[('id', 'i'), ('equal', 'equal'), ('numero', '2')]

Prueba 3:

lexer(“i equal: 2”)

**ValueError**: Token invalido

Esto devido a que no existe automata que acepte “:”