Universidad Nacional Autónoma de México
Facultad de Ciencias
Compiladores
2020-1
Análisis léxico
Prof. Adrian Ulises Mercado Martínez
Ayudante. Carlos Gerardo Acosta Hernández

Objetivo:

Elaborar el analizador léxico como un módulo que pueda ser conectado posteriormente con el resto del compilador.

Desarrollo:

Para la siguiente gramática:

1. program → declaraciones funciones
2. declaraciones → tipo lista_var **\n** declaraciones | **registro inicio** declaraciones **fin \n** | ε
3. tipo → base tipo_arreglo
4. base → **ent** | **real** | **dreal** | **car** | **sin**
5. tipo_arreglo → **[num]** tipo_arreglo | ε
6. lista_var → lista_var **␣ id** | **id**
7. sentencia → sentencia **\n** sentencia | **si** expresion_booleana **entonces \n** sentencia **\n fin \n** | **si** expresion_booleana **\n** sentencia **\n sino \n** sentencia **\n fin \n** | **mientras \n** expresion_booleana **hacer \n** sentencia **\n fin \n** | **hacer \n** sentencia **\n mientras que** expresion_booleana | **id :=** expresion | **escribir** expresion | **leer** variable | **devolver** | **devolver** expresion
8. expresion_booleana → expresion_booleana **oo** expresion_booleana | expresion_booleana **yy** expresion_booleana | **no** expresion_booleana | expresion relacional expresion | **verdadero** | **falso**
9. relacional → < | > | <= | >= | == | <>
10. expresion → expresion **+** expresion | expresion **-** expresion | expresion ***** expresion | expresion **/** expresion | expresion **%** expresion | **(**expresion **)** | variable | **num** | **cadena** | **caracter** | **id(** parametros **)**
11. param_arr → **id [ ]** | param_arr **[ ]**
12. variable → **id** parte_arreglo | **id.id**
13. parte_arreglo → **[** expresion **]** parte_arreglo | ε
14. parametros → lista_param | ε
15. lista_param → lista_param **,** param | param
16. param → **id** | param_arr
17. funciones → **func** tipo **id(** argumentos **) inicio \n** sentencia **\n fin \n** funciones | ε
18. argumentos → listar_arg | **sin**
19. lista_arg → lista_arg arg | arg
20. arg → tipo **id**

    (a) Separar en terminales y no terminales.

(b) En base del conjunto de terminales generar la expresiones regulares para reconocerlos.

(c) Con las expresiones regulares hacer un programa en lex que permita retornar solo las clases léxicas como un número entero para cada diferente tipo de token.

(d) Además el analizador léxico debe reconocer dos tipos de comentarios

   i. De una sola línea que comienza con $--$

   ii. Multilínea que comienza con $< *$ y termina con $* >$ sin que exista entre esos símbolos un $* >$