Compiladores
Análisis Léxico
Programa 1

A. **Descripción del problema.** Se presenta la siguiente gramática para la cual se requiere hacer un análisis léxico, separando los terminales de los no terminales, realizando las expresiones regulares de cada uno de los terminales así como el AFD resultante de este análisis. Una vez terminado dicho análisis se deberá programar el analizador léxico utilizando flex.

----*sin: significa sin tipo, car: tipo caracter*-----
1. programa → declaraciones funciones
2. declaraciones → tipo lista_var**;** declaraciones | tipo_registro lista_var**;** declaraciones | ε
3. tipo_registro → **estructura inicio** declaraciones **fin**
4. tipo → base tipo_arreglo
5. base → **ent** | **real** | **dreal** | **car** | **sin**
6. tipo_arreglo → **[num]** tipo_arreglo | ε
7. lista_var → lista_var**, id** | **id**
8. funciones → **def** tipo **id(**argumentos**) inicio** declaraciones sentencias **fin** funciones | ε
9. argumentos → lista_arg | **sin**
10. lista_arg → lista_arg**,** arg | arg
11. arg → tipo_arg **id**
12. tipo_arg → base param_arr
13. param_arr → **[ ]** param_arr | ε
14. sentencias → sentencias sentencia | sentencia
15. sentencia → **si** e_bool **entonces** sentencia **fin**
| **si** e_bool **entonces** sentencia **sino** sentencia **fin** | **mientras** e_bool **hacer** sentencia **fin**
| **hacer** sentencia **mientras** e_bool**;** | **segun (**variable**) hacer** casos predeterminado **fin**
| variable **:=** expresion **;** | **escribir** expresion **;** | **leer** variable **;** | **devolver;**
| **devolver** expresion**;** | **terminar;** | **inicio** sentencias **fin**
16. casos → **caso num:** sentencia casos | **caso num:** sentencia
17. predeterminado → **pred:** sentencia | ε
18. e_bool → e_bool **o** e_bool | e_bool **y** e_bool | **no** e_bool | **(** e_bool **)** | relacional | **verdadero** | **falso**
19. relacional → relacional oprel relacional | expresion
20. oprel → > | < | >= | <= | <> | =
21. expresion → expresion oparit expresion | expresion **%** expresion | **(**expresion **)** | **id**
| variable | **num** | **cadena** | **caracter** | **id(**parametros**)**
22. oparit → + | − | ∗ | /
23. variable → dato_est_sim | arreglo

24. dato_est_sim → dato_est_sim **.id** | **id**
25. arreglo → **id [** expresion **]** | arreglo **[** expresion **]**
26. parametros → lista_param | ε
27. lista_param → lista_param**,** expresion | expresion

B. **Diseño de la solución**

I. Separar los terminales de los no terminales

Terminales		**No terminales**	
estructura	sino	programa	oprel
inicio	mientras	declaraciones	expresion
fin	hacer	tipo_registro	oparit
ent	segun	tipo	variable
real	escribir	base	dato_est_sim
dreal	leer	tipo_arreglo	arreglo
car	devolver	lista_var	parametros
sin	terminar	funciones	lista_param
num	caso	argumentos	
id	pred	lista_arg	
def	o	arg	

( )	y	tipo_arg	
[ ]	no	param_arr	
,	verdadero	sentencias	
;	falso	sentencia	
.	%	casos	
:	cadena	predeterminado	
si	caracter	e_bool	
entonces		relacional	

II. Las expresiones regulares para los terminales

caracter-> [a-zA-Z]+

num-> [0-9]+

digito->[0-9]

id-> caracter_(digito)*

cadena->(caracter)+

III. El AFD resultante(Imagen)

C. Implementación. Describir cómo está implementado su programa, las partes que lo componen (no es todo el código).

D. Forma de ejecutar el Programa.