Universidad Nacional Autónoma de México

Facultad de Ingeniería

Asignatura: Compiladores

Alumnos:

Carrillo Ruiz Mariana

García Hernández Rogelio

Hernández Carrillo Juan Carlos

Quintero Torres Luis Leonardo

Programa 1

Analizador léxico

Grupo: 2

Fecha de entrega: 26/5/2020

Analisis del problema:

Se necesita crear un analizador léxico para poder reconocer los símbolos de la siguiente gramática:

Gramática

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 → listar_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
    | relacional | **verdadero** | **falso**
19. relacional → relacional > relacional
    | relacional < relacional
    | relacional <= relacional
    | relacional >= relacional
    | relacional <> relacional
    | relacional = relacional
    | expresion
20. expresion → expresion + expresion
    | expresion − expresion
    | expresion * expresion
    | expresion / expresion
    | expresion **%** expresion | **(**expresion **)** |
    | variable | **num** | **cadena** | **caracter**
21. variable → **id** variable_comp
22. variable_comp → dato_est_sim | arreglo | **(** parametros **)**
23. dato_est_sim → dato_est_sim **.id** | ε
24. arreglo → **(** expresion **)** | arreglo **(** expresion **)**
25. parametros → lista_param | ε
26. lista_param → lista_param, expresion | expresion

Se utilizará Lex para crear el analizador Léxico.

Conjunto de los elementos terminales:

terminales
{;,estructura,inicio,fin,ent,real,dreal,car,sin,(,),num,def,tipo,si,entonces,si,sino,mientras,hacer,según,escribir,leer,devolver,terminar,caso,num,pred,:,o,y,no,verdadero,falso,<,>,<=,>=,=,cadena,carácter,+,-,*,/,.,%,id,}

estructura →estructura

inicio → inicio

fin → fin

e→ent

r→real

d→dreal

c→car

s→sin

letra→ [a-zA-z]

cadena→letra*

id→letra(_|letra|digito)*

def→def

oparit [+-=*]

si→si

entonces→entonces

sino→sino

mientras→mientras

hacer→hacer

según→según

escribir→escribir

leer→leer

devolver→devolver

terminar→terminar

caso→caso

verdadero→verdadero

falso→falso

num→ ent | real | dreal

y→y

no→no

Pred → pred

Implementación del programa:

En y.tab.h se definieron nuestras variables, esta cabecera la mandamos a llamar en lexer.l; donde definimos las expresiones regulares.

En main.c se encuentran las funciones de C donde están las funciones para mostrar en pantalla algunos errores como si faltan argumentos y si el archivo no se puede abrir.

En el archivo lexer.l se encuentran nuestras expresiones regulares asi como la especificación de los tokens.

En el archivo prueba.txt se encuentra un ejemplo de una funcion para probar el analizador Lexico.

Forma de ejecución:

Para ejecutar el programa se introducirán los siguientes comandos en la consola de Linux

flex lexer.l

gcc main.c lex.yy.c -o analizadorLexico

./analizadorLexico prueba.txt

Cerradura

Cerradura {

e -> •ent

r -> •real

d-> •dreal

car-> •caracter

num-> •digito+

id-> •letra(_|letra|digito)*

} = q0

Goto(q0,e) = {e•nt} = q1

Goto(q0,r) = {r•eal} = q2

Goto(q0,d) = {d•real} = q3

Goto(q0,c) = {caracter•} = q4

Goto(q0,digito+) = {digito+•

•digito+

} = q5

Goto(q0,letra)= { letra(•_|letra|digito)*

letra(_|•letra|digito)*

letra(_|letra|•digito)*

letra(_|letra|digito)* •

} = q6

Goto(q1, n) = {en•t} = q7

Goto(q2,e) = {re•al} = q8

Goto(q3,r) = {dr•eal} = q9

Goto(q6,_) = {letra(_•|letra|digito)*

letra(_|letra|digito)* •

} = q6

Goto(q7,t) = {ent•} = q10

Goto(q8,a) = {rea•l} = q11

Goto(q9,e) = {dre•al} = q12

Goto(q11,l)= {real•} = q13

Goto(q12,a)={drea•l} = q14

Goto(q14,l)={dreal•} = q15

Goto(q6,digito) = { letra(_|letra|digito•)*

letra•(_|letra|digito)*

letra(•_|letra|digito)*

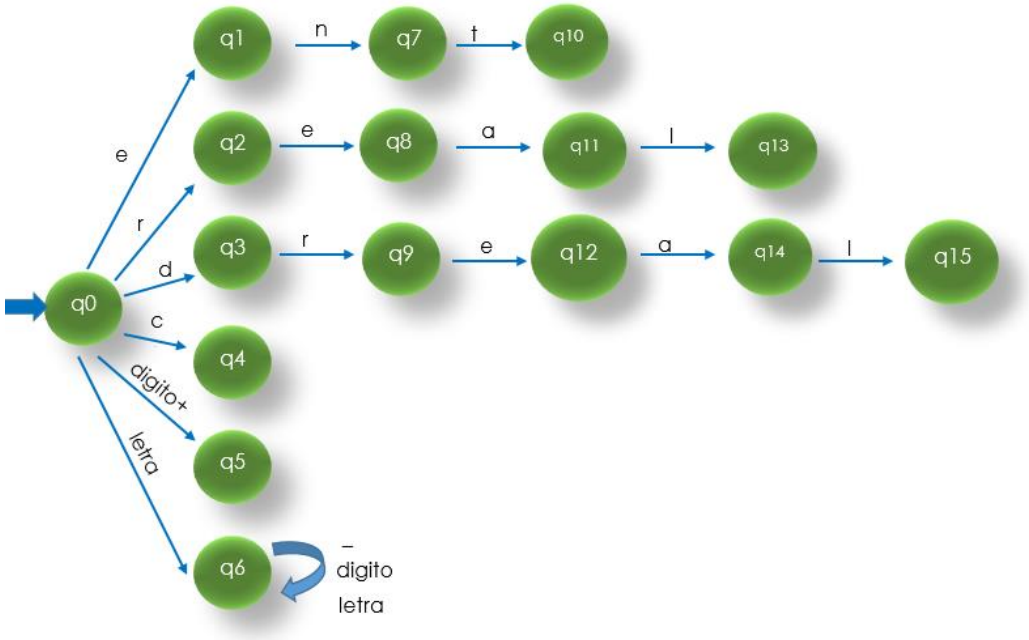
letra(_|•letra|digito)*

letra(_|letra|•digito)*

letra(_|letra|digito)* •

} = q6

AFD


q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10
q11
q12
q13
q14
q15
e
r
d
c
digito+
letra
n
t
e
a
l
r
e
a
l
–
digito
letra