Universidad Nacional Autónoma de México
Semestre 2020-2
Compiladores
Análisis Léxico
Alumnos: González Zepeda Felix
Mendoza Velázquez Daniel Adrián
Ramírez Ancona Simón Eduardo
Valeriano Barrios Cristian
Programa 2

a) Análisis del problema

Con base a la gramática presentada se requiere diseñar un analizador léxico que reconozca a los símbolos terminales, se utilizarán palabras similares a las de la gramática propuesta para detectar errores con mayor facilidad. Se tendrán palabras, números y signos representados como símbolos no terminales que el programa deberá poder retornar como TOKEN's esto lo hará identificando las expresiones regulares.

b) Diseño de la solución

i. Diagramas de sintáxis:

Anexado en archivo PDF adjunto en el repositorio.

ii. En caso de quitar ambigüedad incluir el proceso
iii. En caso de eliminar recursividad incluir el proceso
iv. En caso de factorizar incluir el proceso.

Nota: Se omitieron los pasos ii, iii, iv ya que el orden de precedencia en los símbolos terminales proporcionó una gramática no ambigua, ayudando a que byacc no tuviera errores al momento de procesar la gramática.

c) Implementación. Describir cómo está implementado su programa, las partes que lo componen.

La implementación del análisis sintáctico se hizo por medio del parser.y, programado con la herramienta byacc.

El programa está compuesto por una declaración de todos los tokens que serán reconocidos a partir de la gramática propuesta. Hay una discriminación con los elementos que cuentan con asociatividad izquierda, asociatividad derecha y los que no tienen asociatividad.

Una vez declarada nuestra lista de no terminales, con ellos declaramos nuestro esquema de traducción basado en la gramática propuesta.

Finalmente se utilizó la función yyerror() para depurar los errores detectados en el programa de entrada.

d) Forma de ejecutar el programa.

Para la ejecución del programa se escribió un script de shell (Linux) llamado ***tester.sh*** que compila el lexer, el parser y el main ejecutando la prueba un un archivo llamado prueba. Se anexa el código:

```
#! /bin/bash
flex lexer.l
byacc -d parser.y
gcc lex.yy.c y.tab.c main.c -o main
./main prueba
```

e) Utilizar el analizador léxico obtenido en el primer programa.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
%}

/*Indicar que solo lea un fichero de entrada*/
%option noyywrap
/*contador de lineas*/
%option yylineno

/*Identificadores*/

letra [a-zA-Z]
caracter (\'{letra}\')
digito [0-9]
digitos {digito}+
num  (([1-9]{digito}*)|[0])
exp [eE][+-]?{digitos}
doble ((."{digitos}|{digitos}"."{digitos}){exp}?|{digitos}{exp})
flotante ((."{digitos}|{digitos}"."{digitos}){exp}?|{digitos}{exp})[fF]
id  ({letra}|_)({letra}|{digito}|_)*
cadena \"[^"]*\"

/*Expresiones regulares*/
/* Modificación Simón R. 24/05/20*/
/*Se separan los símbolos terminales para que retornen token individuales.*/
/* Modificación Simón R. 25/05/20*/
/*Agregué expresiones para flotantes y dobles.*/
/*Declaracion de identificadores y correccion de simbolos mal
  Implementados, junto con los comentarios incorrectos Daniel V. 25/05/20*/
%%
"estructura"        { return ESTRUCTURA; }
"inicio"            { return INICIO; }
"fin"               { return FIN; }
"+"                 { return MAS; }
"-"                 { return MENOS; }
"*"                 { return MUL; }
"/"                 { return DIV; }
"%"                 { return MODULO; }
"{"                 { return A_LLAVE; }
"}"                 { return C_LLAVE; }
"("                 { return A_PAR; }
")"                 { return C_PAR; }
"["                 { return A_CORCHETE; }
"]"                 { return C_CORCHETE; }
";"                 { return PYC; }
"ent"               { return ENT; }
```

```
"real"              { return REAL; }
"dreal"             { return DREAL; }
"car"               { return CAR; }
"sin"               { return SIN; }
">"                 { return S_MAYOR; }
"<"                 { return S_MENOR; }
">="                { return S_MAYORIG; }
"<="                { return S_MENORIG; }
"<>"                { return DIFERENTE; }
"="                 { return IGUAL; }
","                 { return COMA; }
"def"               { return DEF;}
"."                 { return PUNTO; }
"si"                { return SI; }
"entonces"          { return ENTONCES; }
"sino"              { return SINO; }
"mientras"          { return MIENTRAS; }
"hacer"             { return HACER; }
"segun"             { return SEGUN; }
"escribir"          { return ESCRIBIR; }
"leer"              { return LEER; }
"devolver"          { return DEVOLVER; }
":="                { return ASIG; }
"terminar"          { return TERMINAR; }
"caso"              { return CASO; }
"pred"              { return PRED; }
":"                 { return D_PUNTOS; }
"o"                 { return OR; }
"y"                 { return AND; }
"no"                { return NOT; }
"verdadero"         { return VERDADERO; }
"falso"             { return FALSO; }
{num}               { return NUM; }
{flotante}          { return NUM; }
{doble}             { return NUM; }
{caracter}          { return CARACTER; }
{cadena}            { return CADENA; }
{id}                { return ID; } /*En caso de encontrar algo del identificador*/
[ \t\n\r]+          {}          /*Para que espacios en blanco sean ignorados*/
.                   {  {printf("Error Lexico: %s\n En línea %d", yytext, yylineno);} }
/*Para los simbolos no reconocidos error lexico*/
%%
```

f) Archivo main.c para ejecutar el programa.

Este es el archivo main.c principal pero no es el que ejecuta el programa completo para eso se creo un script de linux llamado **tester.sh** el cual se encuentra en el inciso d).

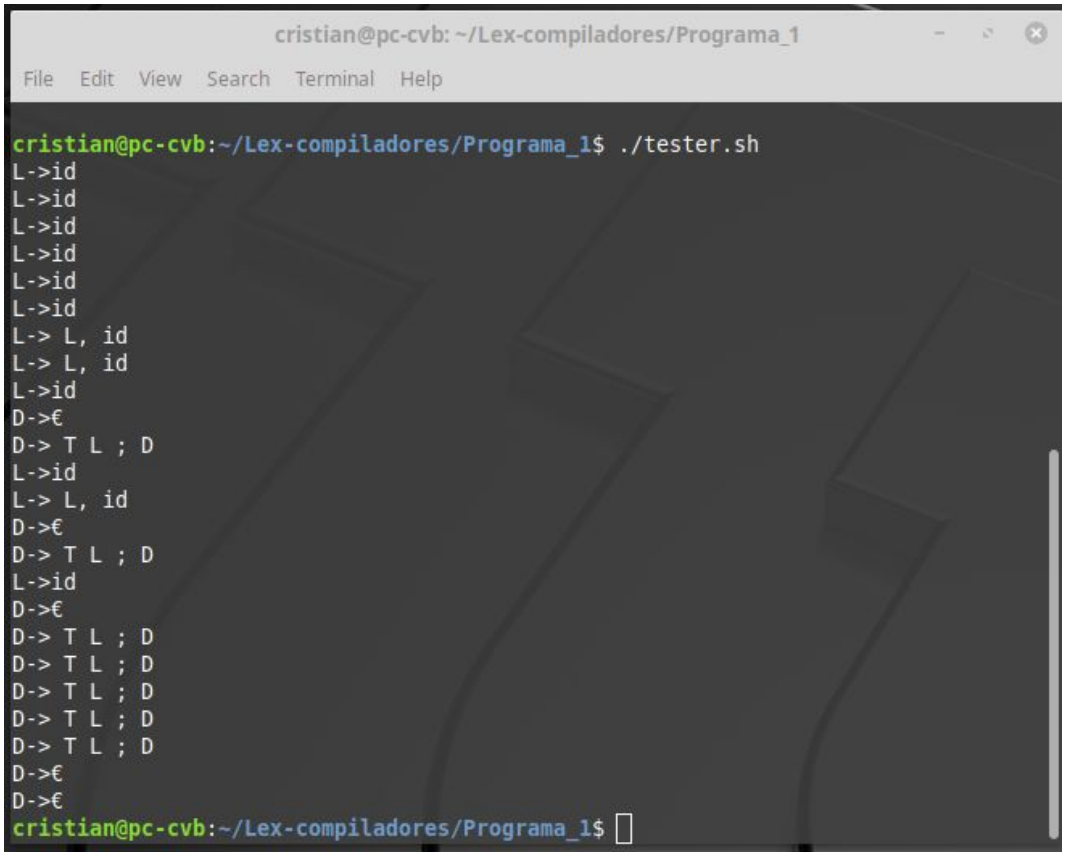
```
/* Creación Simon R 26/05/20 */
#include <stdio.h>

extern int yyparse();
extern FILE *yyin;

int main(int argc, char** argv){
   FILE *f;
   if(argc < 2){
       printf("Faltan argumentos\n");
       return -1;
```

```
    }
    f = fopen(argv[1],"r");
    if(!f){
        printf("El archivo %s no se puede abrir\n",argv[1]);
        return -1;
    }
    yyin = f;
    yyparse();
    fclose(yyin);
}
```

Archivo prueba

```
ent x;
real ya;
dreal de;
car e;
ent [56][25]d;

estructura inicio
    sin arriba, abajo1_, salir;
    estructura inicio
        real huy;
    fin variable_1, variable_3 ;
fin variable_2;

def sin _test(sin) inicio
    x := 12;
    e := 'q';
    e := 'y';
    escribir "Hola mundo";

    terminar;
fin

def real test2(dreal numero1, car df ) inicio
    si verdadero y falso entonces
        inicio
            e := 'a' ;
            de := 4.0e3 ;
            ya := 78.0e-6f ;

        fin
    sino
        devolver (5 % 6) ;
    fin
fin
```

Ejecución


cristian@pc-cvb: ~/Lex-compiladores/Programa_1
File Edit View Search Terminal Help
cristian@pc-cvb:~/Lex-compiladores/Programa_1$ ./tester.sh
L->id
L->id
L->id
L->id
L->id
L->id
L-> L, id
L-> L, id
L->id
D->€
D-> T L ; D
L->id
L-> L, id
D->€
D-> T L ; D
L->id
D->€
D-> T L ; D
D-> T L ; D
D-> T L ; D
D-> T L ; D
D-> T L ; D
D->€
D->€
cristian@pc-cvb:~/Lex-compiladores/Programa_1$