



10 DE NOVIEMBRE DE 2020

INSTITUTO POLITECNICO NACIONAL

Escuela superior de cómputo.

Ingeniería en sistemas computacionales.


PRÁCTICA 4

ANALIZADOR LÉXICO LENGUAJE C

ERICK ALAN MANZANO NAVA

PROFESOR: RAFAEL NORMAN SAUCEDO DELGADO

GRUPO: 3CV6



Contenido

INTRODUCCION.....	2
Desarrollo.	3
Metodología.	3
Ejemplificando el lenguaje.....	3
Identificar las clases léxicas.	4
Expresiones regulares para las clases léxicas.....	5
Código en Lex.	7
Pruebas.....	10
Conclusiones.....	10
Referencias.....	10
Anexos.	11

INTRODUCCION.

En el siguiente documento se mostrará cómo se desarrolla un analizador léxico de lenguaje C, el cual tiene como principio desarrollar un compilador de lenguaje c a ensamblador, se eligió este proyecto ya que es algo que se puede utilizar mucho en microcontroladores.

Se deducirán las clases léxicas y expresiones regulares, para el analizador léxico del lenguaje c aplicando como base el generador de analizadores léxicos llamado Flex en su versión 2.6.4.

Desarrollo.

Metodología.

Ejemplificando el lenguaje.

Ejemplo 1.

- Inclusión de bibliotecas, declaraciones de variables y tipos, y
- secuencia de funciones
- Una y sólo una función denominada main (programa principal)
- Función:
- tipoRetorno Nombre (parámetros) {sentencias}

```
/* ejemplo 1.- Escribe un mensaje en pantalla */  
#include <stdio.h> /* incluye biblioteca donde se define E/S */  
int main()  
{  
/*Este comentario es ignorado por el compilador y*/  
/*no genera código */  
printf("\nIntroducción a la programación en lenguaje C");  
return 0;  
}
```

Ejemplo 2.

- Declaración variable:
- tipo Nombre [=valor]
- Asignación:
- variable = expresión

```
/* ejemplo 2.- multiplica dos números enteros y muestra el resultado por pantalla */  
#include <stdio.h>  
int main()  
{  
int multiplicador; /*se define multiplicador como un entero */  
int multiplicando; /*se define multiplicando como un entero */  
int res; /*se define resultado como un entero*/  
multiplicador = 1000; /*se asignan valores*/  
multiplicando=2;  
res=multiplicador*multiplicando;  
printf("Resultado =%d",res); /*se muestra el resultado */  
return 0;  
}
```

Ejemplo 3.

```
/* ejemplo 3.- se realizan conversiones de tipos implícitas y explícitas */
#include <stdio.h>
int main() {
double d, e, f = 2.33 ;
int i = 6 ;
e = f * i ; /* e es un double de valor 13.98 */
printf( "Resultado = %f", e);
d = (int) ( f * i ) ; /* d es un double de valor 13.00 */
printf( "Resultado = %f", d);
d = (int) f * i ; /* f se ha convertido a un entero truncando */
/* sus decimales, d es un double de valor 13.00 */
printf( "Resultado = %f", d);
return 0;
}
```

Identificar las clases léxicas.

Se identificaron las siguientes clases léxicas.

- Palabra reservada.
- Tipo de dato.
- Ciclo.
- Condicional.
- ID
- Constante hexadecimal.
- Constante long int.
- Constante double.
- Flotante.
- Cadena.
- Asignaciones.
- Operadores.
- Corrimientos.
- Saltos de línea.

Expresiones regulares usando Lex de Flex.

5

```

"/=" { printf("--OPERADOR_DIVISION--"); }
"%=" { printf("--OPERADOR_MODULO--"); }
"&=" { printf("--OPERADOR_AND--"); }
"^=" { printf("--OPERADOR_XOR--"); }
"|"= { printf("--OPERADOR_OR--"); }
">>" { printf("--DESPLAZAMIENTO A LA DERECHA--"); }
"<<" { printf("--DESPLAZAMIENTO A LA IZQUIERDA--"); }
"++" { printf("--OPERADOR DE INCREMENTO--"); }
"--" { printf("--OPERADOR DE DECREMENTO--"); }
">" { printf("--PTR_OP--"); }
"&&" { printf("--AND_OP--"); }
"||" { printf("--OR_OP--"); }
"<=" { printf("--LE_OP--"); }
">=" { printf("--GE_OP--"); }
"==" { printf("--EQ_OP--"); }
"!=" { printf("--NE_OP--"); }
"." { printf("--;--"); }
("{ "|" "<% ") { printf("--{--"); }
("}" "|" "%>") { printf("--}--"); }
"," { printf("--,--"); }
":" { printf("--:--"); }
"_" { printf("-- OPERADOR _--"); }
 "(" { printf("--(--"); }
 ")" { printf("--)--"); }
("[ "|" "<:") { printf("--[--"); }
("]" "|" ">:") { printf("--]--"); }
"." { printf("--. --"); }
"&" { printf("-- OPERADOR LOGICO --"); }
"!" { printf("-- OPERADOR LOGICO --"); }
"~" { printf("-- OPERADOR LOGICO --"); }
"-_" { printf("-- - - -"); }
"+" { printf("-- OPERADOR --"); }
"*" { printf("-- OPERADOR --"); }
"/" { printf("-- OPERADOR --"); }
"%" { printf("-- OPERADOR --"); }
"<" { printf("--<--"); }
">" { printf("-->--"); }
"^" { printf("-- OPERADOR LOGICO--"); }
"|" { printf("-- OPERADOR LOGICO --"); }
"?" { printf("--OPERADOR LOGICO--"); }

. { /* ignore bad characters */ }

```

Código en Lex.

```
% { /* definiciones de lenguaje C */
    /* lenguaje C */

#include<stdio.h>

/*
    Los analizadores léxico se definen con expresiones regulares.
    Las expresiones regulares son una notación simplificada
    de conjuntos regulares.
    Los conjuntos regulares: conjuntos finitos + operaciones
    operaciones: unión, concatenación, cerraduras ( +, *)
*/
% }

D          [0-9]
L          [a-zA-Z_]
H          [a-zA-F0-9]
E          [Ee][+-]?{D}+
FS         (f|F|l|L)
IS         (u|U|l|L)*

%%

"auto"     { printf("--AUTO-- PALABRA RESERVADA"); }
"break"    { printf("--BREAK-- PALABRA RESERVADA"); }
"case"     { printf("--CASE-- PALABRA RESERVADA"); }
"char"     { printf("--CHAR-- TIPO DE DATO"); }
"const"    { printf("--CONST-- PALABRA RESERVADA"); }
"continue" { printf("--CONTINUE-- PALABRA RESERVADA"); }
"default"  { printf("--DEFAULT-- PALABRA RESERVADA"); }
"do"       { printf("--DO-- CICLO"); }
"double"   { printf("--DOUBLE-- TIPO DE DATO"); }
"else"     { printf("--ELSE-- CONDICIONAL"); }
"enum"     { printf("--ENUM-- PALABRA RESERVADA"); }
"extern"   { printf("--EXTERN-- PALABRA RESERVADA"); }
"float"    { printf("--FLOAT-- TIPO DE DATO"); }
"for"      { printf("--FOR-- CICLO"); }
"go to"    { printf("--GOTO-- PALABRA RESERVADA"); }
"if"       { printf("--IF-- CONDICIONAL"); }
"int"      { printf("--INT-- TIPO DE DATO"); }
"long"     { printf("--LONG-- TIPO DE DATO"); }
"register" { printf("--REGISTER-- PALABRA RESERVADA"); }
```



```

"return"          { printf("--RETURN-- PALABRA RESERVADA"); }
"short"           { printf("--SHORT-- TIPO DE DATO"); }
"signed"          { printf("--SIGNED-- PALABRA RESERVADA"); }
"sizeof"          { printf("--SIZEOF-- PALABRA RESERVADA"); }
"static"          { printf("--STATIC-- PALABRA RESERVADA"); }
"struct"          { printf("--STRUCT-- PALABRA RESERVADA"); }
"switch"          { printf("--SWITCH-- CONDICIONAL"); }
"typedef"         { printf("--TYPEDEF-- PALABRA RESERVADA"); }
"union"           { printf("--UNION-- PALABRA RESERVADA"); }
"unsigned"        { printf("--UNSIGNED-- TIPO DE DATO"); }
"void"            { printf("--VOID-- PALABRA RESERVADA"); }
"volatile"        { printf("--VOLATILE-- PALABRA RESERVADA"); }
"while"           { printf("--WHILE-- CICLO"); }

{L}({L}|{D})*      { printf("--ID--"); }

0[xX]{H}+{IS}?     { printf("--CONSTANTE HEXADECIMAL--"); }
0{D}+{IS}?         { printf("--IDENTIFICADOR--"); }
{D}+{IS}?          { printf("--LONG INT--"); }
L?'(\\.|[^\"])+' { printf("--CADENA --"); }

{D}+{E}{FS}?       { printf("-- CONSTANTE --"); }
{D}*"."{D}+({E})?{FS}? { printf("--FLOTANTE--"); }
{D}+"."{D}*({E})?{FS}? { printf("--double--"); }

L?"(\\.|[^\"])*"    { printf("--STRING_LITERAL--"); }

"..."            { printf("--ELLIPSIS--"); }
">=>"             { printf("--ASIGNACION A LA DERECHA--"); }
"<=<="            { printf("--ASIGNACION A LA IZQUIERDA--"); }
"+="              { printf("--OPERADOR_SUMA--"); }
"-= "             { printf("--OPERADOR_RESTA--"); }
"*="              { printf("--OPERADOR_MULTIPLICACION--"); }
"/="              { printf("--OPERADOR_DIVISION--"); }
"%="              { printf("--OPERADOR_MODULO--"); }
"&="              { printf("--OPERADOR_AND--"); }
"^="              { printf("--OPERADOR_XOR--"); }
"|="              { printf("--OPERADOR_OR--"); }
">>"              { printf("--DESPLAZAMIENTO A LA DERECHA--"); }
"<<"              { printf("--DESPLAZAMIENTO A LA IZQUIERDA--"); }
"++"              { printf("--OPERADOR DE INCREMENTO--"); }
"--"              { printf("--OPERADOR DE DECREMENTO--"); }
">-"              { printf("--PTR_OP--"); }
"&&"              { printf("--AND_OP--"); }
"||"              { printf("--OR_OP--"); }
"<="              { printf("--LE_OP--"); }
">="              { printf("--GE_OP--"); }

```

```

"=="          { printf("--EQ_OP--"); }
"!=="        { printf("--NE_OP--"); }
";"          { printf("--;--"); }
("{|" "<% ") { printf("--{--"); }
("}"|" ">") { printf("--}--"); }
";"          { printf("--,--"); }
":"          { printf("--:--"); }
"="          { printf("-- OPERADOR --"); }
"("          { printf("--(--"); }
")"          { printf("--)--"); }
("[|" "<:") { printf("--[--"); }
("]"|" ">") { printf("--]--"); }
"."          { printf("--. --"); }
"&"          { printf("-- OPERADOR LOGICO --"); }
"!"          { printf("-- OPERADOR LOGICO --"); }
"~"          { printf("-- OPERADOR LOGICO --"); }
"_"          { printf("-- - --"); }
"+"          { printf("-- OPERADOR --"); }
"*"          { printf("-- OPERADOR --"); }
"/"          { printf("-- OPERADOR--"); }
"% "         { printf("-- OPERADOR --"); }
"<"          { printf("--<--"); }
">"          { printf("-->--"); }
"^"          { printf("-- OPERADOR LOGICO--"); }
"|"          { printf("-- OPERADOR LOGICO --"); }
"? "         { printf("--OPERADOR LOGICO--"); }

.            { /* ignore bad characters */ }

%%

```

Pruebas.

Después de compilar el código con un makefile nos pide que metamos valores para que nos regrese la clase léxica de dicha expresión regular.

```
erick@erick-VGN-CR420E:~/Escritorio/Compis$ make run
gcc -c main.c
main.c: In function 'main':
main.c:3:2: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
  yylex(); /* < inicia el autómata */
  ^~~~~~
flex lexico.l
gcc -c lex.yy.c
gcc main.o lex.yy.o -lfl
./a.out
hola
--ID--
j
--ID--
++
--OPERADOR DE INCREMENTO--
--
--OPERADOR DE DECREMENTO--
+=
--OPERADOR_SUMA--
int x = 5;
--INT-- TIPO DE DATO--ID---- OPERADOR ----LONG INT----;--
F=0X00;
--ID---- OPERADOR ----CONSTANTE HEXADECIMAL----;--
INT incremento = 5.12;
--ID----ID---- OPERADOR ----FLOTANTE----;--
int inc = 4.21;
--INT-- TIPO DE DATO--ID---- OPERADOR ----FLOTANTE----;--
```

Conclusiones.

Esta practica me gusto mucho ya que me voy dando cuenta de todo lo que hay detrás de un compilador, se me hizo fácil la práctica, pero un poco confusa en la parte de entender que es lo que se pedía hacer, me falta un poco de conocimiento para poder entender los significados como “clases léxicas”. Hablando de flex es una herramienta muy buena que nos ayuda a generar el análisis léxico.

Referencias.

Infor.uva.es. n.d. *Ejemplos De Programas En C*. [online] Available at: <https://www.infor.uva.es/~fdiaz/so/2004_05/doc/SO_PR01_20041026> [Accessed 10 November 2020].

Lysator.liu.se. 1995. *ANSI C Grammar (Lex)*. [online] Available at: <<https://www.lysator.liu.se/c/ANSI-C-grammar-l.html>> [Accessed 10 November 2020].

Quiñones Hernández, G. (2018). GENERADOR DE ANALIZADORES LÉXICOS (LEX & FLEX). Retrieved 10 November 2020, from <https://prezi.com/p/skykfvjrxm7/generador-de-analizadores-lexicos-lex-flex/>

Anexos.

Makefile, para la compilación de la practica se tiene que tener el makefile dentro de la carpeta junto con el main.c y el lexico.l

Solo en consola se da la instrucción make run.

```
lex.yy.c: lexico.l
    flex lexico.l

lex.yy.o: lex.yy.c
    gcc -c lex.yy.c

main.o: main.c
    gcc -c main.c

a.out: main.o lex.yy.o
    gcc main.o lex.yy.o -lfl

clean:
    rm -f main.o lex.yy.o a.out lex.yy.c

run: a.out
    ./a.out
```

Main.c

```
int main(void) {

    yylex(); /* < inicia el autómata */

    return 0;
}
```