

Reporte de laboratorio: Analizador Léxico para lenguaje BASIC

Sebastián de Jesús Marroquín Martínez

October 2019

1 Introducción

La tarea principal de un *analizador léxico o scanner* es dividir un flujo de entrada de caracteres en strings (secuencia de caracteres) con significado propio (tokens).

Lex es una utilidad para la generación automática de los analizadores léxicos.

Para esto, describimos algunos puntos importantes sobre los Analizadores léxicos:

1. El analizador léxico *convierte en tokens* el flujo (stream) de entrada.
2. Los Tokens son los símbolos *terminales* de un lenguaje, por ejemplo: ingles, español, lenguajes de programación, etc.
3. Las expresiones regulares definen los *símbolos terminales ó tokens*.

2 Teoría de Lex

La entrada a Lex se divide en tres secciones con el símbolo %% dividiendo las secciones.

La entrada se copia para generar un carácter a la vez. Siempre se requiere el primer %, ya que siempre debe haber una sección de reglas. Sin embargo, si no especificamos ninguna regla, la acción predeterminada es hacer coincidir todo y copiarlo en la salida.

Puede usarse de ejemplo lo siguiente:

```
%%  
/* match everything except newline */  
. ECHO;  
/* match newline */  
\n ECHO;  
%%  
int yywrap(void) {  
    return 1;  
}  
int main(void) {  
    yylex();  
    return 0;  
}
```

Como se puede ver en el ejemplo anterior, se han especificado dos patrones en la sección de reglas. Cada patrón debe comenzar en la columna uno. Esto es seguido por espacios en blanco (espacio, tabulación o nueva línea) y una acción opcional asociada con el patrón. La acción puede ser una sola declaración C o varias declaraciones C, encerradas entre llaves. Cualquier cosa que no comience en la columna uno se copia literalmente al archivo C generado.

La sección de definiciones se compone de sustituciones, código y estados de inicio. El código en la sección de definiciones simplemente se copia como está en la parte superior del archivo C generado y debe estar entre corchetes con marcadores "% {" y "%}". Las sustituciones simplifican las reglas de coincidencia de patrones. Por ejemplo, podemos definir dígitos y letras de la siguiente manera:

```
digit [0-9]  
letter [A-Za-z]  
%{  
    int count;  
}%  
%%  
/* match identifier */  
{letter}({letter}|{digit})* count++;  
%%  
int main(void) {  
    yylex();  
    printf("number of identifiers = %d\n", count);  
    return 0;  
}
```

Los espacios en blanco deben separar el término definitorio y la expresión asociada. Las referencias a las sustituciones en la sección de reglas están rodeadas por llaves (letra) para distinguirlas de los literales. Cuando tenemos una coincidencia en la sección de reglas, se ejecuta el código C asociado.

3 Análisis de la Tarea

3.1 El lenguaje BASIC

Acorde con [1]: *"BASIC es un lenguaje de programación que se creó con fines pedagógicos, era el lenguaje que utilizan las microcomputadoras de los años 80. Actualmente sigue siendo muy conocido y tienen muchísimos dialectos muy diferentes al original. Veamos algo de historia del Basic. En los años 1960, las computadoras tenían un valor elevado y se usaban para tareas específicas, eran mono tarea. Pero luego, permitiendo que algunas empresas pequeñas pudieran permitirse adquirirlas. Las computadoras mejoraron mucho en velocidad, capacidad de procesamiento de datos y también lo hicieron los lenguajes. Aparecieron lenguajes como el FORTRAN, el COBOL y el Basic."*

En [2], la primera versión de BASIC tenía 14 comandos, todos con nombres directos y sintaxis que tenían sentido:

1. **PRINT**: Imprime texto y números de salida (y, más tarde, lo muestra en las pantallas de terminales de tiempo compartido y PC).
2. **LET**: Realiza cálculos y asigna el resultado a una variable.
3. **IF & THEN**: permiten que el programa determine si una declaración fue verdadera.
4. **FOR & NEXT**: permiten que un programa se ejecute en bucles.
5. **GOTO**: deja que un programa se bifurque a otra línea numerada dentro de sí mismo.
6. **END**: Llega a la conclusión del programa.

Aparte de todos estos comandos, se encuentra el comando **INPUT**, un comando que permitía que un programa BASIC aceptara caracteres alfanuméricos escritos por un usuario.

3.2 Solución de la Tarea


Se anexa a continuación una captura de pantalla del código implementado para resolver esta tarea.

```
ana_lex.l
1  digito    [0-9]
2  digitos   {digito}+
3  numero    {digitos}(.{digitos})?(E{digitos})?
4  letra     [A-Za-z]
5  maymen    [<|>]
6  oparit    [-|+|*|/]
7  oprel     =(2){({maymen})(-)?}
8  id        {letra}(_|{letra}){digitos}*
9  desc      .
10 lin       [\n]
11 asig      \:=
12 porcentaje %
13 %{
14
15 #include <stdlib.h>
16 #include <stdio.h>
17
18 int yycolval = 1;
19
20 %}
21
22 %%
23 LET      {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <LET> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
24 INPUT    {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <INPUT> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
25 PRINT    {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <PRINT> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
26 IF       {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <IF> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
27 THEN     {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <THEN> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
28 ELSE     {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <ELSE> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
29 FOR      {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <FOR> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
30 NEXT     {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <NEXT> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
31 {numero} {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <numero> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
32 {id}      {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <id> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
33 {oparit}  {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <oparit> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
34 {oprel}   {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <oparel> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
35 {asig}    {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <asignacion> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
36 {lin}     {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <linea> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
37 [\t]     {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <tabulador> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
38 \"       {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <comillas> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
39 \{       {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <llave_abre> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
40 \}       {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <llave_cierra> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
41 ;        {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <punto_y_coma> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
42 %        {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <Porcentaje> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
43 {desc}    {fprintf(yyout, "FILA: %d COLUMNA: %d TOKEN: <desconocido> LEXEMA: <%s>\n", yylineno, yycolval+=yyleng, yytext);}
44 %%
45
46 int yywrap(void) {
47     return 1;
48 }
49
50 int main(int argc, char **argv)
51 /*int argc;
52 char** argv;*/
53 {
54     FILE *leer;
55     leer = fopen(argv[1], "r");
56     if (!leer){
57         printf("No se encontro %s\n", argv[1]);
58         exit(1);
59     }
60     yyin = leer;
61
62     FILE *escribir = fopen("tokens.txt", "w");
63     if (escribir == NULL){
64         printf("Error creando %s\n", "tokens.txt");
65         exit(1);
66     }
67     yyout = escribir;
68     yylex();
69     fclose(escribir);
70     return 0;
71 }
72
```

El método para compilar el archivo se muestra a continuación:

```
sebastian@DESKTOP-M1LRMKV: /mnt/c:/Users/sebas/Desktop/UAM/19-P/Traductores/Workshop/v02 - BASIC - LexicalAnalyzer/02$ lex ana_lex.l
sebastian@DESKTOP-M1LRMKV: /mnt/c:/Users/sebas/Desktop/UAM/19-P/Traductores/Workshop/v02 - BASIC - LexicalAnalyzer/02$ cc lex.yy.c -lfl -lm
sebastian@DESKTOP-M1LRMKV: /mnt/c:/Users/sebas/Desktop/UAM/19-P/Traductores/Workshop/v02 - BASIC - LexicalAnalyzer/02$ ./a.out data.txt
sebastian@DESKTOP-M1LRMKV: /mnt/c:/Users/sebas/Desktop/UAM/19-P/Traductores/Workshop/v02 - BASIC - LexicalAnalyzer/02$
```

El archivo de prueba que se ocupo para las pruebas de nuestro analizador léxico es el siguiente:

 data: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
% LET variable = expression
% IF condition THEN statement
% ELSE IF condition THEN statement
% FOR
% statements
% NEXT
% INPUT valores
```

3.3 Resultados Obtenidos

La salida del analizador léxico construido se presenta a continuación:

```
tokens: Bloc de notas
Archivo Edición Formato Ver Ayuda
FILA: 1 COLUMNA: 2 TOKEN: <Porcentaje> LEXEMA: <%>
FILA: 1 COLUMNA: 3 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 6 TOKEN: <LET> LEXEMA: <LET>
FILA: 1 COLUMNA: 7 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 15 TOKEN: <id> LEXEMA: <variable>
FILA: 1 COLUMNA: 16 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 17 TOKEN: <desconocido> LEXEMA: <=>
FILA: 1 COLUMNA: 18 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 28 TOKEN: <id> LEXEMA: <expression>
FILA: 1 COLUMNA: 29 TOKEN: <desconocido> LEXEMA: <
>
FILA: 1 COLUMNA: 30 TOKEN: <slinea> LEXEMA: <
>
FILA: 1 COLUMNA: 31 TOKEN: <Porcentaje> LEXEMA: <%>
FILA: 1 COLUMNA: 32 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 34 TOKEN: <IF> LEXEMA: <IF>
FILA: 1 COLUMNA: 35 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 44 TOKEN: <id> LEXEMA: <condition>
FILA: 1 COLUMNA: 45 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 49 TOKEN: <THEN> LEXEMA: <THEN>
FILA: 1 COLUMNA: 50 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 59 TOKEN: <id> LEXEMA: <statement>
FILA: 1 COLUMNA: 60 TOKEN: <desconocido> LEXEMA: <
>
FILA: 1 COLUMNA: 61 TOKEN: <slinea> LEXEMA: <
>
FILA: 1 COLUMNA: 62 TOKEN: <Porcentaje> LEXEMA: <%>
FILA: 1 COLUMNA: 63 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 67 TOKEN: <ELSE> LEXEMA: <ELSE>
FILA: 1 COLUMNA: 68 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 70 TOKEN: <IF> LEXEMA: <IF>
FILA: 1 COLUMNA: 71 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 80 TOKEN: <id> LEXEMA: <condition>
FILA: 1 COLUMNA: 81 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 85 TOKEN: <THEN> LEXEMA: <THEN>
FILA: 1 COLUMNA: 86 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 95 TOKEN: <id> LEXEMA: <statement>
FILA: 1 COLUMNA: 96 TOKEN: <desconocido> LEXEMA: <
>
FILA: 1 COLUMNA: 97 TOKEN: <slinea> LEXEMA: <
>
FILA: 1 COLUMNA: 98 TOKEN: <Porcentaje> LEXEMA: <%>
FILA: 1 COLUMNA: 99 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 102 TOKEN: <FOR> LEXEMA: <FOR>
FILA: 1 COLUMNA: 103 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 104 TOKEN: <desconocido> LEXEMA: <
>
FILA: 1 COLUMNA: 105 TOKEN: <slinea> LEXEMA: <
>
FILA: 1 COLUMNA: 106 TOKEN: <Porcentaje> LEXEMA: <%>
FILA: 1 COLUMNA: 107 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 117 TOKEN: <id> LEXEMA: <statements>
FILA: 1 COLUMNA: 118 TOKEN: <desconocido> LEXEMA: <
>
FILA: 1 COLUMNA: 119 TOKEN: <slinea> LEXEMA: <
>
FILA: 1 COLUMNA: 120 TOKEN: <Porcentaje> LEXEMA: <%>
FILA: 1 COLUMNA: 121 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 125 TOKEN: <NEXT> LEXEMA: <NEXT>
FILA: 1 COLUMNA: 126 TOKEN: <desconocido> LEXEMA: <
>
FILA: 1 COLUMNA: 127 TOKEN: <slinea> LEXEMA: <
>
FILA: 1 COLUMNA: 128 TOKEN: <Porcentaje> LEXEMA: <%>
FILA: 1 COLUMNA: 129 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 134 TOKEN: <INPUT> LEXEMA: <INPUT>
FILA: 1 COLUMNA: 135 TOKEN: <desconocido> LEXEMA: < >
FILA: 1 COLUMNA: 142 TOKEN: <id> LEXEMA: <valores>
```

References

- [1] Lenguaje de Programación BASIC. (2019). Retrieved 5 October 2019, from <http://www.larevistainformatica.com/BASIC.htm>
- [2] MCCRACKEN, H. (2019). <https://time.com>. Retrieved 6 October 2019, from <https://time.com/69316/basic/>