

# Elaboración de un analizador lexicográfico con Flex

CARLOS AYALA T.<sup>1</sup>, WILSON RAMOS B.<sup>1</sup>, DENNIS VEINTIMILLA A.<sup>1</sup>

1. Ingeniería en Sistemas Informáticos y de Computación, Facultad de Sistemas, Escuela Politécnica Nacional, Quito, Ecuador.

**Abstract**—Compilers are structured by many analyzers, in this case we need to develop a lexical analyzer. This analyzer was created using some references and was develop using Flex tool, which provide the proper lexical analyzer. After the creation of the lexical analyzer, it's important to have in consideration the errors that may appear, most of them should be corrected to finish the analyzer. This errors are common and easy to detect, most of them should appear in early develop phases, but some of them may appear at the end of the project, however we should know how to fix them.

**Index Terms**—compilers, errors, lexical analyzer.

**Resumen**—Los compiladores están estructurados por muchos analizadores, en este caso necesitamos desarrollar un analizador léxico. Este analizador fue creado usando algunas referencias y fue desarrollado usando la herramienta Flex, la cual provee un analizador léxico apropiado. Luego de la creación del analizador léxico, es importante considerar los errores que pueden aparecer, muchos de ellos deben ser corregidos para finalizar el analizador. Estos errores son comunes y fáciles de detectar, muchos de ellos pueden aparecer en etapas tempranas de desarrollo, pero muchos otros pueden aparecer al final, sin embargo, se debe conocer cómo arreglarlos.

**Términos de Indexación**—compiladores, errores, analizador léxico.

## I. INTRODUCCIÓN

LOS programas, en la actualidad, ocupan gran parte de la red, haciendo que la convivencia con dichos programas sea constante. Estos programas están estructurados en lenguajes de programación, y a su vez un lenguaje de programación consta de un compilador que reconoce los códigos escritos y los lleva a un programa. Un compilador entonces, tendrá ciertos criterios de control tales como analizadores del tipo léxico, semántico y morfológico, los cuales serán aplicados para corregir errores en los códigos.

Un analizador léxico, opera bajo petición del analizador sintáctico devolviendo así un componente léxico conforme al

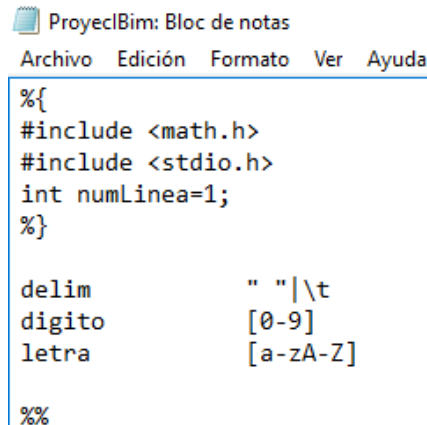
analizador sintáctico va necesitando [1]. Con esta premisa surge la necesidad de crear un analizador léxico, el cual utiliza Flex como estrategia de creación del analizador. Flex es una herramienta usada para generar escáneres los cuales son programas que reconocen ciertos patrones léxicos en un texto [2].

Es así, que el objetivo principal es usar la herramienta Flex, para la creación de un archivo en c y posteriormente de un ejecutable que será el analizador léxico. Todo esto definiendo previamente las reglas que utilizara dicho analizador y también definiendo errores en la compilación de un texto o código.

## II. MATERIALES Y MÉTODOS

En primer lugar, para el desarrollo del analizador léxico, se debió conocer su estructura, esto se logró en base a ejemplos, además de conocer que la implementación de la herramienta Flex se la podía llevar a cabo en Windows y Linux.

Como primer paso se realizó la declaración de las



```
%{
#include <math.h>
#include <stdio.h>
int numlinea=1;
%}

delim      " "\t
digito     [0-9]
letra      [a-zA-Z]

%%
```

Fig. 1. Definición de las reglas básicas del analizador, así como ciertas librerías que serán usadas por el Flex.

expresiones regulares con las cuales trabajaría nuestro analizador.

Una vez declaradas las expresiones regulares del compilador se procedieron a buscar ejemplos de la declaración de las reglas del compilador [3].

Las reglas que un compilador tiene dependerán del que lo esté diseñando, es por esto que un compilador es único para cada lenguaje, ya que se define un conjunto totalmente diferente de reglas. En esta caso las reglas que se debían

utilizar estaban especificadas para que no exista ambigüedad en su aplicación, así mismo se dejaba un espacio para la creatividad de los desarrolladores.

Con esta premisa se procedió a la implementación de las

**1. Palabras reservadas del lenguaje, que serán las siguientes:**

int, float, bool, char, string, if, then, else, while, do, input, output, return

## 2. Caracteres especiales:

, ; : ( ) [ ] { } + - \* / < > = ! & \$

### 3. Operandos compuestos:

```
<=> == != && ||
```

#### 4. Identificadores:

Los identificadores del lenguaje son utilizados para designar nombres de variables y funciones

### 5. Literales:

Los literales son formas de describir constantes en código fuente.

- Literales de tipo int son representados como repeticiones de uno o más dígitos, anteceditos opcionalmente por un signo menos (-).
- Literales en float son formados con un entero seguido de punto decimal y una secuencia de dígitos decimales.
- Literales de tipo bool pueden ser true o false.
- Literales de tipo char son representados por un único caracter entre comillas simples como 'a', '+' o '='.
- Literales de tipo string son cualquier secuencia de caracteres entre comillas dobles como "mi nombre" o "x = 3"; y sirven apenas para imprimir mensajes con el comando output.

Fig. 2. Consideraciones de las reglas básicas que se utilizarán en el proyecto.

reglas en el analizador léxico, las cuales serían la base del reconocimiento de los patrones en un texto.

Una vez definidas las reglas básicas que llevarían a cabo el

```
{delim)+
"bool"      {printf(" Palabra clave: %s\n", yytext);}
"chan"     {printf(" Palabra clave: %s\n", yytext);}
"do"       {printf(" Palabra clave: %s\n", yytext);}
"else"     {printf(" Palabra clave: %s\n", yytext);}
"float"    {printf(" Palabra clave: %s\n", yytext);}
"if"       {printf(" Palabra clave: %s\n", yytext);}
"input"    {printf(" Palabra clave: %s\n", yytext);}
"int"      {printf(" Palabra clave: %s\n", yytext);}
"output"   {printf(" Palabra clave: %s\n", yytext);}
"return"   {printf(" Palabra clave: %s\n", yytext);}
"string"   {printf(" Palabra clave: %s\n", yytext);}
"then"     {printf(" Palabra clave: %s\n", yytext);}
"while"    {printf(" Palabra clave: %s\n", yytext);}

"true"|"false" {printf(" Boolean: %s\n", yytext);}
[digit0]*".". [digit0]*([+|-]?[digit0]+)? [ ]? [digit0]+
{printf(" Decimal: %g\n", atof(yytext));}
[ ]? [digit0]+
{printf(" Entero: %d\n", atoi(yytext));}
[letre]*([letre]*[digit0]*)*?
{printf(" Identificador: %s\n", yytext);}
"["([a-z][A-Z][0-9]" ]*[a-z][A-Z][0-9]" ]*)"
{printf(" String: %s\n", yytext);}
"["([A-Z][0-9]" ]*[a-z][A-Z][0-9]" ]*)"
{printf(" Char: %s\n", yytext);}

"."|"["|"]"|"{"|"}"|"("|")"|"["|"]"|"["|"]"
{printf(" Caracteres especiales: %s\n", yytext);}
"["|"]"|"{"|"}"|"("|")"|"["|"]"|"["|"]"|"&"|"%"
{printf(" Caracteres especiales: %s\n", yytext);}
"<"|">"|"="|"!="|"&&"|"||"
{printf(" Operador Compuesto: %s\n", yytext);}

"/"|"*"|"%"|"^"|"["|"]"|"["|"]"|"["|"]"
{printf("\n");
numLinea=numLinea+1; printf("%d ", numLinea);}
/* Si en nuestra entrada tiene algún caracter que no pertenece a las reglas anteriores, se genera un error léxico
{printf(" Error léxico en línea: %d de %s\n", numLinea, yytext);}
```

Fig. 3. Reglas implementadas en el archivo lex, que componen la base fundamental del analizador.

control del analizador se debía incluir las instrucciones de usuario [4].

Las instrucciones de usuario son aquellas instrucciones adicionales que dependen del creador del analizador en este caso, se definió como instrucción de usuario, un contador para la ejecución del analizador.

```
%%  
  
int main(void) {  
    printf("%d ", numLinea);  
    yylex();  
}
```

Fig. 4. Instrucciones de usuario, que permitirá contar las líneas en el analizador.

Una vez finalizada la declaración de todas las reglas del analizador, surgía la pregunta del sistema en el cual se llevaría a cabo. Para este caso se eligió Windows como sistema operativo, por lo que se procedería a la instalación de Flex y paralelamente de Bison [5].

Posteriormente y luego de la instalación rápida de Flex, se tenían que configurar las variables de entorno añadiendo la dirección de instalación del Flex a Path.

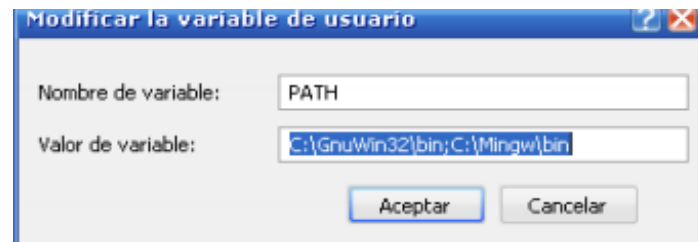


Fig. 5. Variable de entorno necesaria para la ejecución de Flex.

Una vez hecho eso, teníamos ya la herramienta Flex lista, ahora comenzaría la creación del archivo. l, esto guardando el txt en el cual se definieron las reglas del analizador, con dicha extensión. Luego se abrió un command prompt window en modo administrador para poder realizar la transformación apropiada de .l a un archivo c.

Esta acción sería llevada a cabo mediante el comando en Windows flex “nombre\_del\_archivo”.l, con lo cual se tendría un archivo en c.

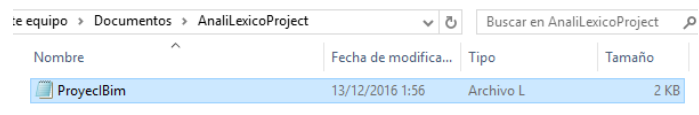


Fig. 6A. Creación del archivo .l

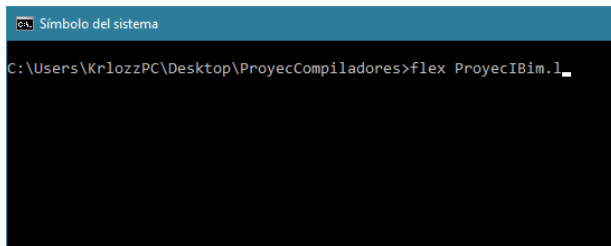


Fig. 6B. Creación del archivo c, mediante el comando de Windows.

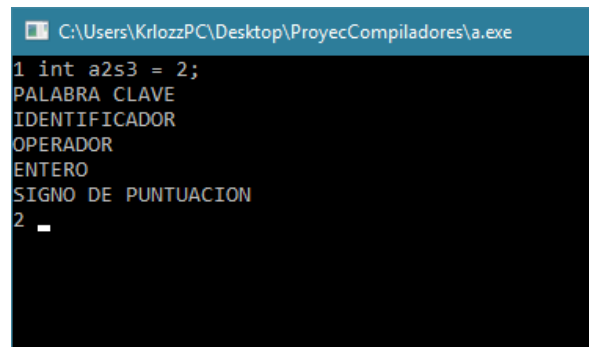


Fig. 8B. Ejemplo

Nombre	Fecha de modifica...	Tipo
lex.yy	13/12/2016 2:09	C source fil
ProyecIBim	13/12/2016 1:56	Archivo L

Fig. 6C. Archivo resultante llamado por el Flex como lex.yy.

Ahora en la misma pantalla de comandos se creará el ejecutable, esto con la instrucción en Windows `gcc lex.yy.c -lfl` la cual transforma el archivo en c a un ejecutable .exe en Windows .out en Linux, que básicamente guarda la información de nuestro analizador.

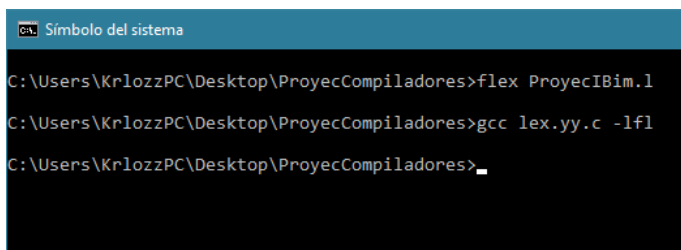


Fig. 7A. Creación del ejecutable mediante el comando en Windows.

Nombre	Fecha de modifica...	Tipo
a	13/12/2016 2:09	Aplicación
lex.yy	13/12/2016 2:09	C source file
ProyecIBim	13/12/2016 1:56	Archivo L

Fig. 7B. Resultado de la creación del ejecutable, a.exe.

Una vez creado el exe, prácticamente tenemos nuestro analizador listo, solo hace falta ejecutarlo y escribir programas ejemplos para que el analizador pueda identificar, tokens, variables, caracteres y errores del tipo léxico.

### III. RESULTADOS Y DISCUSION

A continuación, se detalla un ejemplo con el analizador léxico en ejecución.

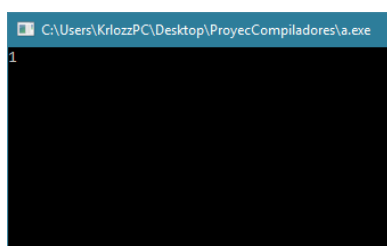


Fig. 8A. Ejemplo

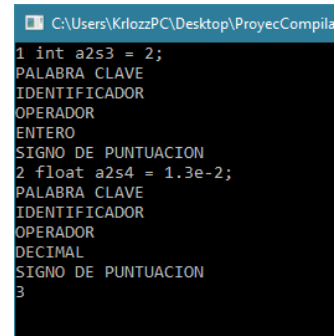


Fig. 8C. Ejemplo

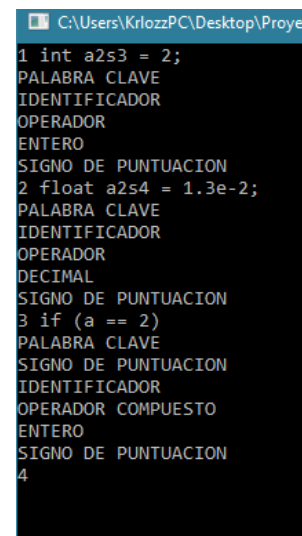


Fig. 8D. Ejemplo

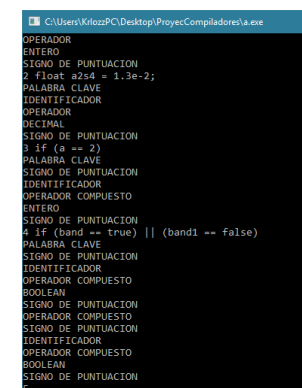


Fig. 8E. Ejemplo

```

C:\Users\KrllozPC\Desktop\ProyecCompiladores\
1 output("Hola Mundo");
PALABRA CLAVE
SIGNO DE PUNTUACION
STRING
SIGNO DE PUNTUACION
SIGNO DE PUNTUACION
2

```

Fig. 9A. Ejemplo 2

```

C:\Users\KrllozPC\Desktop\ProyecCompiladores\
1 input: a = '12';
PALABRA CLAVE
SIGNO DE PUNTUACION
IDENTIFICADOR
OPERADOR
CHAR
SIGNO DE PUNTUACION
2 _

```

Fig. 9B. Ejemplo 2

```

C:\Users\KrllozPC\Desktop\ProyecCompiladores\
1 input: a = '12';
PALABRA CLAVE
SIGNO DE PUNTUACION
IDENTIFICADOR
OPERADOR
CHAR
SIGNO DE PUNTUACION
2 bool band;
PALABRA CLAVE
IDENTIFICADOR
SIGNO DE PUNTUACION
3

```

Fig. 9C. Ejemplo 2

```

C:\Users\KrllozPC\Desktop\ProyecCompiladores\
1 input: a = '12';
PALABRA CLAVE
SIGNO DE PUNTUACION
IDENTIFICADOR
OPERADOR
CHAR
SIGNO DE PUNTUACION
2 bool band;
PALABRA CLAVE
IDENTIFICADOR
SIGNO DE PUNTUACION
3 while (as1 >= 12)
PALABRA CLAVE
SIGNO DE PUNTUACION
IDENTIFICADOR
OPERADOR COMPUESTO
ENTERO
SIGNO DE PUNTUACION
4

```

Fig. 9D. Ejemplo 2

```

C:\Users\KrllozPC\Desktop\ProyecCompiladores\
1 input: a = '12';
PALABRA CLAVE
SIGNO DE PUNTUACION
IDENTIFICADOR
OPERADOR
CHAR
SIGNO DE PUNTUACION
2 bool band;
PALABRA CLAVE
IDENTIFICADOR
SIGNO DE PUNTUACION
3 while (as1 >= 12)
PALABRA CLAVE
SIGNO DE PUNTUACION
IDENTIFICADOR
OPERADOR COMPUESTO
ENTERO
SIGNO DE PUNTUACION
4 int a ? 2-
PALABRA CLAVE
IDENTIFICADOR
ERROR LEXICO EN LA LINEA: 4 DE ?
ENTERO
OPERADOR
5

```

Fig. 9E. Ejemplo 2

```

C:\Users\KrllozPC\Desktop\ProyecCompiladores\
1 input: a = '12';
PALABRA CLAVE
SIGNO DE PUNTUACION
IDENTIFICADOR
OPERADOR
CHAR
SIGNO DE PUNTUACION
2 bool band;
PALABRA CLAVE
IDENTIFICADOR
SIGNO DE PUNTUACION
3 while (as1 >= 12)
PALABRA CLAVE
SIGNO DE PUNTUACION
IDENTIFICADOR
OPERADOR COMPUESTO
ENTERO
SIGNO DE PUNTUACION
4 int a ? 2-
PALABRA CLAVE
IDENTIFICADOR
ERROR LEXICO EN LA LINEA: 4 DE ?
ENTERO
OPERADOR
5 #3 < 7
ERROR LEXICO EN LA LINEA: 5 DE #
ENTERO
OPERADOR
ENTERO
6

```

Fig. 9F. Ejemplo 2

En los distintos ejemplos se puede observar cómo trabaja el analizador en base a las instrucciones que se le dio, identificando enteros, palabras claves, signos de puntuación entre otros.

También tenemos ejemplos de errores donde el analizador los detecta de manera correcta, lo cual demuestra la correcta implementación del analizador [6].

#### IV. CONCLUSIONES

En la aplicación de este analizador se pudo observar la importancia fundamental en la definición de las reglas básicas del compilador, sin las cuales no podría identificar los tokens ingresados. Las instrucciones de usuario, en este caso cumplen la función de enumeración y da un toque personal a la implementación y desarrollo del analizador.

Destacable es la diferencia entre la implementación en Windows o Linux, en los cuales cambiaría la forma de instalación, comandos y ejecución, pero conservando básicamente la esencia de creación de los archivos con sus respectivas extensiones.

Se concluye entonces, que el analizador tiene en su esencia básica de compilación las reglas básicas que definimos que son las más importantes, dejando a las de usuario en segundo plano. El analizador en este caso con dichas reglas reconocerá los diferentes tokens, y como ya se habló anteriormente, para cada lenguaje de programación le corresponde un analizador léxico con sus reglas diferentes.

El proyecto se encuentra en GitHub en el siguiente URL: <https://github.com/Krlozz/AnaliLexicoProject>

[ Carlos Ayala, Wilson Ramos, Dennis Veintimilla, 6 «GitHub,» 2016. [En línea]. Available: ] <https://github.com/Krlozz/AnaliLexicoProject>.

#### Biografías



**Carlos Antonio Ayala Tipán**, nació el 3 de Octubre de 1995 en Quito – Ecuador, hijo de Marcelino Ayala y Narciza Tipán. Comenzó sus estudios en la Escuela Unidad Educativa Municipal Experimental “Antonio José de Sucre”, sus siguientes estudios los cursó en el Colegio Experimental e ISPED “Juan Montalvo”, y actualmente realiza sus estudios superiores en la Escuela

Politécnica Nacional, en donde cursa la carrera de Ingeniería en Sistemas Informáticos y de Computación, donde se encuentra en cuarto semestre. Sus hobbies favoritos son incursionar en tópicos de actualidad referente a su carrera, en sus tiempos libres le apasiona practicar algunos deportes tales como realizar bicicleta o el fútbol.

#### REFERENCIAS

- [ F. J. C. Limón, «Galeon.com,» 2015. [En línea]. Available: 1 <http://10380054.galeon.com/u5.htm>. ]
- [ V. Paxson, «TLDP,» Abril 1995. [En línea]. Available: 2 <http://es.tldp.org/Manuales-LuCAS/FLEX/flex-es-2.5.html#SEC2>. ]
- [ J. Ezpeleta, «Universidad de Saragoza,» 2013. [En línea]. 3 Available: ] <http://webdiis.unizar.es/~ezpeleta/lib/exe/fetch.php?media=misdatos:comp2bis.introflex.pdf>.
- [ D. J. B. Torralvo, «Universidad Politécnica de Madrid,» 4 Junio 2014. [En línea]. Available: ] [http://oa.upm.es/32288/1/PFC\\_JOSE\\_BARBERA\\_TORRALVO.pdf](http://oa.upm.es/32288/1/PFC_JOSE_BARBERA_TORRALVO.pdf).
- [ S. Autor, «Tutos.com,» 2013. [En línea]. Available: 5 <http://www.unse-prog2.comxa.com/downloads/Tutorial%20de%20Instalacion%20y%20utilizacion%20de%20Flex-bison.pdf>.



**Dennis Fernando Veintimilla Alvarado**

Nació el 23 de agosto de 1996 en Quito - Ecuador, hijo de Cristóbal Veintimilla y Marianita de Jesús, es el tercero de sus hermanos. Comenzó sus estudios en La Rioja, España en el Colegio Nuestra Señora de la Vega. Estuvo en el “Instituto Ciudad de Haro” institución en la cual alcanzo el título de bachiller científico-tecnológico. En la universidad ingreso a la Escuela Politécnica Nacional para estudiar ingeniería en sistemas informáticos y de computación. Actualmente cursa el cuarto semestre de dicha carrera. Como pasatiempo le gusta jugar fútbol, salir con amigos y aprender cada vez más sobre las nuevas tecnologías.



**Wilson Gabriel Ramos Bravo**, nació el 28 de noviembre de 1996 en Quito - Ecuador, hijo de Wilson Ramos y Lupe Bravo, es el tercero de sus hermanos. Comenzó sus estudios en la Escuela Fiscal Mixta “Gonzalo Zaldumbide” en la cual obtuvo el honor de llegar a ser abanderado del pabellón Nacional. Estuvo en el “Instituto Nacional

Mejía” institución en la cual alcanzo el mérito de ser segundo escolta del pabellón Nacional del Ecuador. En la universidad ingreso a la Escuela Politécnica Nacional para seguir su sueño que está en la ingeniería en sistemas informáticos y de computación. Actualmente cursa el cuarto semestre de dicha carrera. Como pasatiempo le gusta toca la guitarra, jugar

videojuegos, salir con amigos y aprender un poquito más del medio que le rodea. Su visión está planteada en utilizar sus conocimientos en el ámbito astronómico, en el cual Ecuador tiene falencias o poca preocupación.