[[1]](#footnote-2)

Analizador Léxico de un Compilador.

Integrante1: Malan Aucay Silvana Elizabeth

e-mail: silvana.malan@epn.edu.ec

Integrante 2: Tayupanta Bohórquez José

e-mail: jose.tayupanda@epn.edu.ec

*Index Terms*— Compilador, token, expresiones regulares, error léxico, librería Lex, analizador léxico.

# introduccion

En la actualidad para desarrollar un Compilador poseemos de diferentes herramientas que han facilitado la creación de cada etapa. Entre estas herramientas disponemos de Flex y Lex que se encargaran de generar analizadores léxicos y sintácticos en un lenguaje de programación determinado.

Para esto el punto de partida será definir una gramática que sería un conjunto de reglas que nos ayudarían a validar una frase. El objetivo de crear un analizador léxico es generar programas que teniendo como punto de partida una gramática, sea capaz de decidir si una frase está correctamente construida o no en base a dicha gramática.

# DESARROLLO DE CONTENIDOS

## COMPILADOR

Es un programa que lee un programa escrito en un lenguaje fuente y lo traduce a un programa equivalente en otro lenguaje.

Cuando se va a diseñar un compilador, es necesario especificar:

a) el lenguaje fuente

b) el lenguaje objeto

c) el sistema operativo sobre el que funcionará

d) el lenguaje usado para construirlo

La construcción de un compilador involucra la división del proceso en una serie de fases que variará con su complejidad. Generalmente estas fases se agrupan en dos tareas: el análisis del programa fuente y la síntesis del programa objeto.

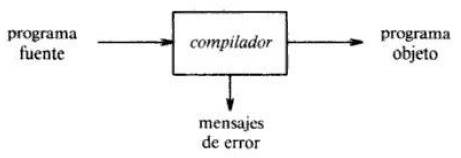
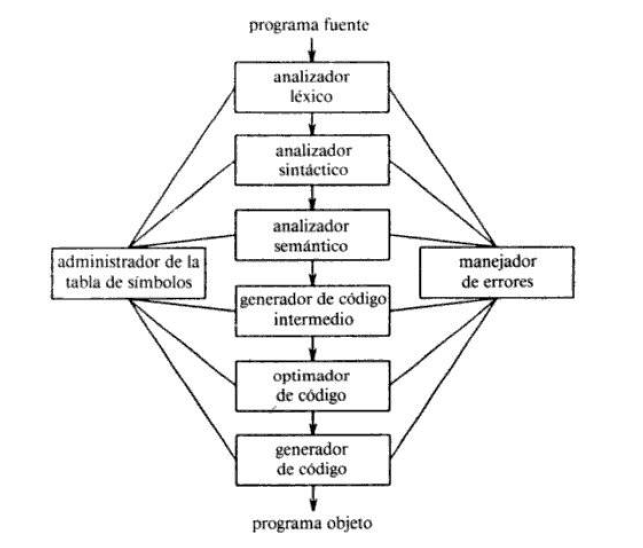


FIG 1. UN COMPILADOR

## FASES DE UN COMPILADOR.

Un Compilador opera en fases, cada una transforma el código de una representación a otra, estas se denominan analizador léxico, analizador sintáctico, analizador semántico, generador de código y optimizador de código.



## ANALIZADOR LÉXICO

Es la primera fase de un compilador y su función principal consiste en leer los caracteres de entrada o mejor conocido como el código fuente y elabora como salida una secuencia de componentes léxicos que pasan al analizador sintáctico.

El analizador léxico ignora tabuladores, espacios en blanco, cambios de línea, comentarios, etc.

Entre las ventajas principales tenemos:

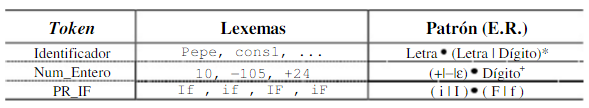
* Separa el análisis léxico del análisis sintáctico.
* Mejora la eficiencia del compilador
* Mejora la transportabilidad

ESPECIFICACIONES DE UN ANALIZADOR LEXICO

**Tokens.**- son los elementos léxicos del lenguaje mientras que para el resto de las fases de un compilador son los símbolos terminales de la gramática

**Patrón.-** expresión regular que define el conjunto de cadenas que puede representar a cada uno de lostokens.

**Lexema.-**secuencia de caracteres del código fuente que concuerda con el patrón de un token.



Cuando el analizador léxico encuentra un lexema devuelve como información a qué token pertenece y todo lo que sabe de él, incluido el propio lexema. En el último caso, se supone que esa palabra pertenece a un lenguaje en el que mayúsculas y minúsculas son equivalentes.

**ERRORES LÉXICOS**

Son escasos los errores que se pueden detectar en el análisis léxico ya que tiene una visión muy restringida del código fuente. Por ejemplo.

Al ingresar la palabra fi, el analizador léxico la tomara como un identificador mas no como una palabra reservada, este análisis deberá hacerlo alguna de las fases siguientes que componen un compilador.

Entre los métodos principales para implementar un analizar léxico tenemos las siguientes:

* Se puede usar un analizador Lex, para producir un analizador léxico a partir de una especificación basada en expresiones regulares.

**Ventaja:** comodidad y rapidez en el desarrollo. **Desventaja**: ineficiencia del analizador resultante y complicado mantenimiento del código generado

* Escribir un analizador en lenguaje de programación, utilizando las diferentes entradas y salidas de este lenguaje para leer la entrada.
* Escribir el analizador en un lenguaje ensamblador y manejar la lectura de entrada.

**Ventaja**: máxima eficiencia y compacidad. **Desventaja**: muy complicado de desarrollar.

En este caso se usó las herramientas de Lex para implantar el analizador léxico, que será descrito a continuación.

## HERRAMIENTA LEX

LEX

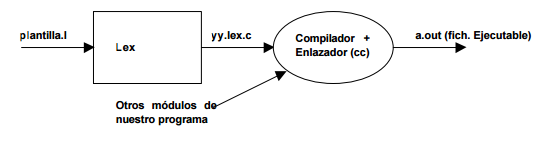
Es un programa que genera analizadores léxicos, fue escrito por Eric Schmidt y Mike Lesk, es el analizador léxico estándar en los sistemas Unix.

**lex.yy.c =** Es el analizador lexicográfico de L generado por Lex; está constituido, en su parte principal, por una función escrita en C que realiza las tareas de análisis lexicográfico basándose en autómatas regulares reconocedores de la forma de las piezas sintácticas de L.

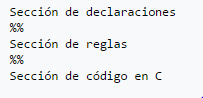
ESTRUCTURA DE LEX.

Internamente Lex va a actuar como un autómata que localizará las expresiones regulares que le describamos, y una vez reconocida la cadena representada por dicha expresión regular, ejecutará el código asociado a esa regla.

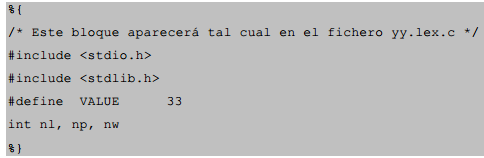
Externamente podemos ver a Lex como una caja negra con la siguiente estructura:



Las expresiones regulares que se incluyen en una especificación Lex definen la forma de las palabras cuya presencia se pretende comprobar en un fichero de tipo texto que se toma como entrada.



La **sección de declaraciones** es el lugar para definir macros y expresiones regulares o patrones y para importar los archivos de cabecera escritos en C.

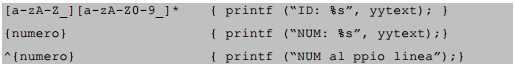


La **sección de reglas** es la sección más importante; asocia patrones a sentencias de C. Los patrones son simplemente expresiones regulares.

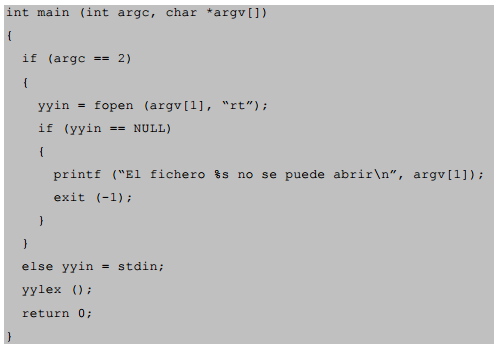
Como normas para la identificación de expresiones regulares, Lex sigue las siguientes:

- Siempre intenta encajar una expresión regular con la cadena más larga posible,

- En caso de conflicto entre expresiones regulares (pueden aplicarse dos o más para una misma cadena de entrada), Lex, se guía por estricto orden de declaración de las reglas.

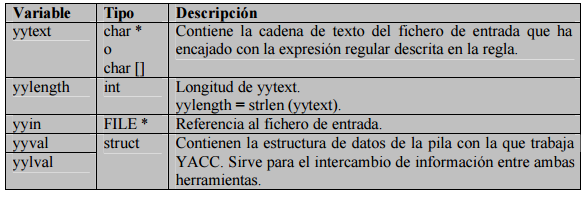


La **sección de código** C contiene sentencias en C y funciones que serán copiadas en el archivo fuente generado.

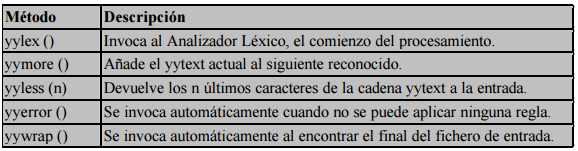


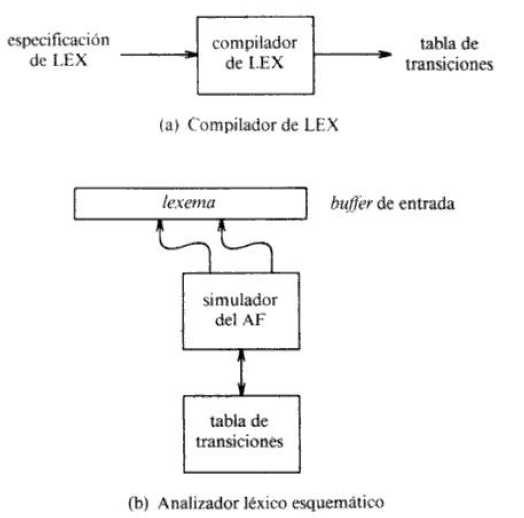
Lex incorpora algunas variables, funciones, procedimientos y macros que nos permiten realizar de una forma más sencilla todo el procesamiento.

Como variables, las más utilizadas son:



Como métodos (funciones y procedimientos), tenemos:





# descripcion del programa.

Para la descripcion del analizador léxico se utilizara los siguientes tokens.

* palabras reservadas

**int, float, double, etc**.

* Identificadores
* Operadores

**\*, +, /, -, %**

* Constantes
* Símbolos de puntuación:

**; , . :**

* Símbolos especiales:

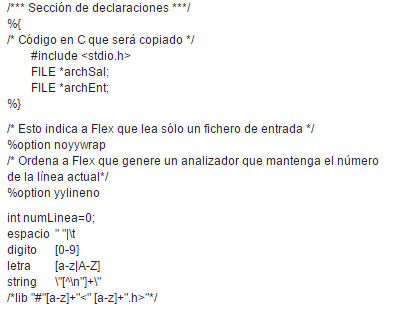
**( ) [ ]**

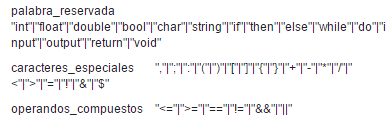
## Gramatica

La gramática es la formación de la jerarquía de las estructuras en los lenguajes de programación; la sintaxis de los lenguajes de programación son suficientemente sencillos o simples para obtener así la eficiencia de los algoritmos

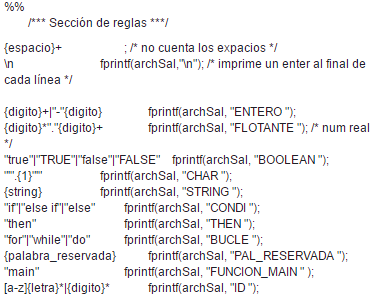
La gramática que se usara para el analizador se encuentra dividida en tres secciones como lo veremos a continuación.

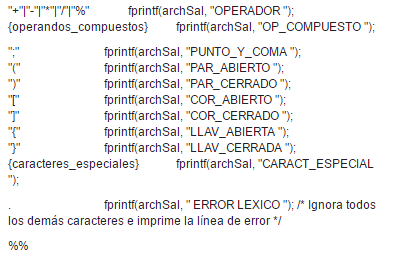
LA SECCIÓN DE DECLARACIONES.





SECCIÓN DE REGLAS.

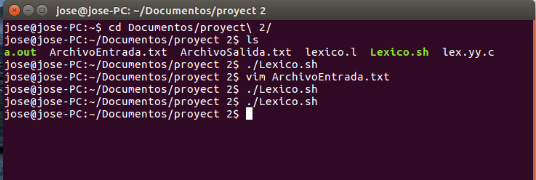




## DESCRIPCION DE FUNCIONAMIENTO

Después de construir la gramática respectiva con las especificaciones necesarias enviamos a correr el programa desde la terminal de Ubuntu.

Le otorgamos los permisos necesarios para que se pueda ejecutar.

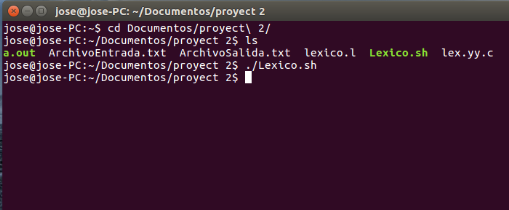


Esta creado un archivo. Sh

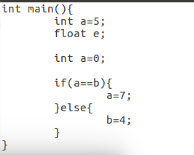
Que se encarga de volver a compilar y tener un ejecutador del archivo léxico.l

Y en ese . sh también está una línea de código que abre el archivo de salida

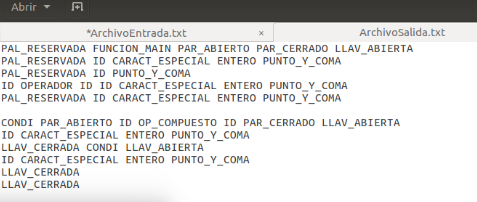
.sh llama a flex, ggc.yy.c



En el archivo de entrada se encuentra las siguientes líneas de código que van a ser analizadas y comparadas a la gramática que construimos.



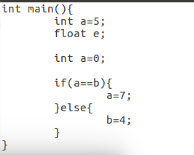
Después de ser analizada el resultado que nos presenta en el archivo de salida nos describe el grupo al que pertenece cada uno de los tokens es el siguiente:



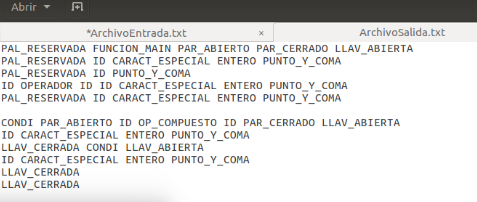
# ANALISIS DE RESULTADOS.

Para comprobar el funcionamiento del analizador léxico vamos a realizar diferentes pruebas y presentar los resultados.

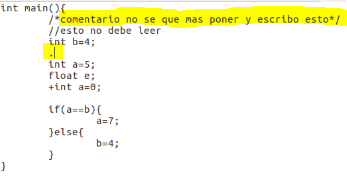
## Si ingresamos el siguiente código el resultado será:



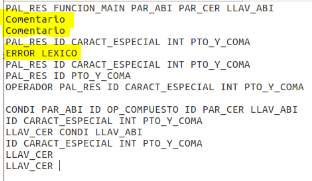
En el archivo de salida nos presenta la descripcion de cada token.



## Si al código le hacemos cambios como son: ingresar un comentario de mayor extensión o un punto.



Nos presentara los siguientes resultados, como que la nueva línea sigue siendo un comentario o el punto se convierte en un error léxico.



El código utilizado y la información correspondiente a las gramáticas se pueden encontrar en el siguiente enlace de GITHUB. <https://github.com/josetabo7/analizador-lexico>

# CONCLUSIONES

* La gramática debe estar correctamente definida y de esta forma evitar ambigüedades, y si tuviéramos una gramática ambigua lo mejor es replantearla por una equivalente.
* Lex puede ser denominado como la primera etapa necesaria a la hora de elaborar un compilador, un intérprete, un emulador o cualquier herramienta que necesite procesar un fichero de entrada para poder cumplir un objetivo.
* El analizador léxico ignora tabuladores, espacios en blanco, cambios de línea, comentarios.
* El trabajo que realiza el analizador léxico no debe poseer falencias al momento de separación de tokens y clasificación ya que esta será la información básica que va ser utilizada por la siguiente fase denominada analizador sintáctico.

References

1. *Sethi, R., & Ullman, J. D. (1998). Compiladores: principios, técnicas y herramientas. Pearson Educación.*
2. [*http://webdiis.unizar.es/~ezpeleta/lib/exe/fetch.php?media=misdatos:compi:2.analisislexico.pdf*](http://webdiis.unizar.es/~ezpeleta/lib/exe/fetch.php?media=misdatos:compi:2.analisislexico.pdf)
3. *SINTÁCTICO, A. L. Y. (2013). Facultad de Ciencias de la Computación.*
4. *Velasco, J. R. (s.f.). Analizador Léxico. Obtenido de* [*https://es.scribd.com/doc/98159365/Analizador-Lexico*](https://es.scribd.com/doc/98159365/Analizador-Lexico)
5. *PUEBLA, B. U. (2013). Visión Histórica del Desarrollo de los Compiladores. Obtenido de http://ymoyao.cs.buap.mx/wp-content/uploads/2013/09/notasCompiladores.pdf*
6. *Lanero, A. V. (s.f.). Una Herramienta para el Análisis Léxico: Lex. Obtenido de* [*https://www.infor.uva.es/~mluisa/talf/docs/aula/A3-A6.pdf*](https://www.infor.uva.es/~mluisa/talf/docs/aula/A3-A6.pdf)
7. [*http://ymoyao.cs.buap.mx/wp-content/uploads/2013/09/notasCompiladores.pdf*](http://ymoyao.cs.buap.mx/wp-content/uploads/2013/09/notasCompiladores.pdf)

**Tayupanta B. Jose**

Nació en Machachi-Ecuador, el 15 de Febrero. Realizo sus estudios secundarios en el colegio “La Salle”, actualmente se desempeña como Estudiante de la Facultad de Ingeniería en Sistemas Informáticos y de Computación de la Escuela Politécnica Nacional.

**Malan A. Silvana**

Nació el 24 de Marzo de 1993 en Cuenca-Ecuador. Sus estudios secundarios los realizo en el Colegio “Santiago de Guayaquil”. Actualmente se encuentra estudiando en la Escuela Politécnica Nacional. En la carrera de Ingeniería en Sistemas Informáticos y de Computación**.**

1. Este trabajo fue realizado el 13-noviembre-2016, por parte de los estudiantes (José Tayupanta y Silvana Malan), estudiantes de la carrera de Ingeniería en Sistemas de la Escuela Politécnica Nacional.

   El presente escrito se fundamenta en la teoría recibida en la materia de Compiladores y Lenguajes, y nos demuestra la creación de un analizador léxico que trabaja con el sistema operativo Linux, y herramientas como son Lex. [↑](#footnote-ref-2)