[[1]](#footnote-1)

Analizador Sintáctico y Tabla de Símbolos

(Febrero 2017)

Caiza Llumitaxi Jonathan Paul, Guzmán Alarcón Darwin Santiago.

**Escuela Politécnica Nacional del Ecuador**

***Resumen***—En este paper se manifiesta la creación del Analizador Sintáctico, con su respectiva tabla de símbolos con la ayuda de la herramienta bison, además de la compilación del analizador lexicográfico y el parser para la creación del ejecutable del compilador, el cual también nos mostrara un informe de errores en caso de una entrada desconocida de código.

***Índice de Términos—*** Analizador Sintáctico, Tabla de Símbolos, Informe de errores, árbol sintáctico abstracto, Ensamblador.

# **Introducción**

Dentro del compilador el texto de entrada ingresado por el usuario pasará al analizador lexicográfico, el cual será el encargado de producir los tokens de dicho texto, los cuales serán utilizados por el analizador sintáctico para determinar la estructura sintáctica del código ingresado a partir de la construcción del árbol sintáctico abstracto apoyándose de la tabla de símbolos. Los datos almacenados en la tabla de símbolos como los identificadores, literales, etc. serán organizados de forma ordenada según su tipo de símbolo.

El compilador en caso de encontrar algún error con entradas desconocidas nos mostrará un informe sobre dicho error para su corrección. También se gestionarán los registros de memoria para cada variable. El lenguaje a analizar por el compilador deberá abarcar el conjunto de variables globales y funciones, además que las declaraciones globales deben terminar con punto y coma.

La creación del parser se basa en la herramienta Bison, además de utilizar lenguaje Ensamblador para la gestión de los registros de memoria para cada variable, ya que el parser se encargada de cargar el contenido de una variable desde la memoria hacia un registro.

El compilador recibirá un archivo “.txt” el cual será el código a compilar, el cual nos presentará la tabla de símbolos y el proceso detallado de la compilación, además de presentarnos los errores con la línea de código y el token correspondientes al error.

# **OBJETIVOS**

***Objetivo general***

* Investigar y diseñar un Analizador Sintáctico con su respectiva Tabla de Símbolos.

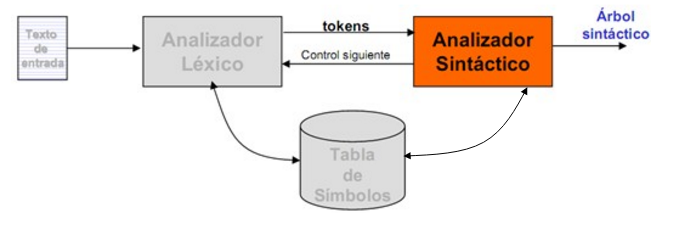
***Objetivo especifico***

* Crear un analizador Sintáctico con la ayuda de la herramienta Bison.
* Generar la tabla de símbolos a partir del texto ingresado por el usuario

# **MARCO TEORICO** [1][2][3]

***Tokens. -*** Representan una unidad de información como son palabras reservadas, secuencia de caracteres, etc.

***Analizador Sintáctico. -*** Esla fase del analizador que se encarga de chequear el texto de entrada en base a una dramática dada. Y en caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce



**Fig. 1** Fase del Analizador Sintáctico

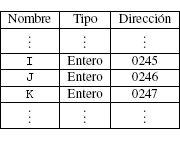
<http://images.slideplayer.es/5/1612715/slides/slide_3.jpg>

***Tabla de símbolos. -***  También se la llama tabla de nombres o tabla de identificadores. La tabla de símbolos lleva el registro de la información sobre el ámbito y el enlace de los nombres y tanto el analizador lexicográfico como el analizador sintáctico examinan la tabla de símbolos cada vez que se encuentra un nombre o identificador en el texto fuente. Si se descubre un nombre nuevo o nueva información sobre un nombre ya existente, se producen cambios en la tabla.

Tiene dos funciones principales:

- Efectuar chequeos semánticos.

- Generación de código



**Fig. 2** Tabla de símbolos

<http://www.mailxmail.com/curso-informatica-ordenador-cientifico-desarrollo/datos-tipos>

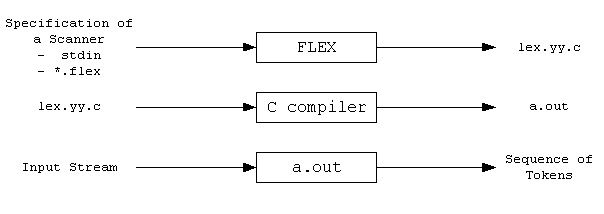
***Expresiones Regulares. –*** Representan patrones que sirven para encontrar una combinación de caracteres determinado dentro de un texto

***Lenguaje Ensamblador. -*** El lenguaje ensamblador es un lenguaje de programación de bajo nivel, y constituye la representación más directa del Código máquina específico para cada arquitectura de computadoras legible por un programador.

# **Materiales y métodos**

## **Herramienta Flex**

Esta herramienta es un generador de scaners que nos reconoce los patrones de un texto según los defínasenos previamente, esta herramienta esta descrita por expresiones regulares y códigos C, al compilar un archivo \*.l este nos genera un nuevo archivo “lex.yy.c”, que será el que compilaremos para generar el ejecutable y así poder usar el analizador lexicográfico.



**Fig. 3** Proceso de generación del analizador lexicográfico con la herramienta flex

<http://alumni.cs.ucr.edu/~lgao/teaching/Img/flex.jpg>

La descripción de la estructura de flex esta expresada de la siguiente forma:

%%

DECLARACIONES

%%

REGLAS DE TRADUCCIÓN

%%

FUNCIONES AUXILIARES

En la parte de las declaraciones se coloca lo que se va usar para la traducción.

%{

/\*DEFINICION DE CONSTANTES\*/

%}

delim [“ ”,\n,\t]

ws {delim}+

letra [a-zA-Z]

num [0-9]

id {letra}({letra}[num])\*

numero {num}+(\.{num}+)?(E[+-]?{num}+)?

.

.

.

En la parte de las reglas de traducción se coloca lo que significa cada token para el compilador.

%%

{ws}

if {return (IF);}

then {return (THEN);}

else {return (ELSE);}

{id} {yyval=(INT) installID (ID)

return (ID);}

{numero} {yyval=(INT) instalNUM(NUMERO)

return (ID);}

“=” {yyval=EQ instalNUM(RELOAP);}

“<” {yyval=LT instalNUM(RELOAP);}

“<=” {yyval=LE instalNUM(RELOAP);}

.

.

.

En la parte de las funciones auxiliares se coloca las funciones necesarias q se van a utilizar.

%%

Int installID(){

}

Int installNUM(){

}

Int main(){

}

## **Herramienta Bison**

Esta herramienta es un generador de analizadores sintácticos con extensión “.y” que convierte que se convierte en un documento en C que será el encargado de analizas la gramática.

La descripción de la estructura de bison esta expresada de la siguiente forma:

%{

DECLARACIONES

%}

DECLARACIONES DE BISON

%%

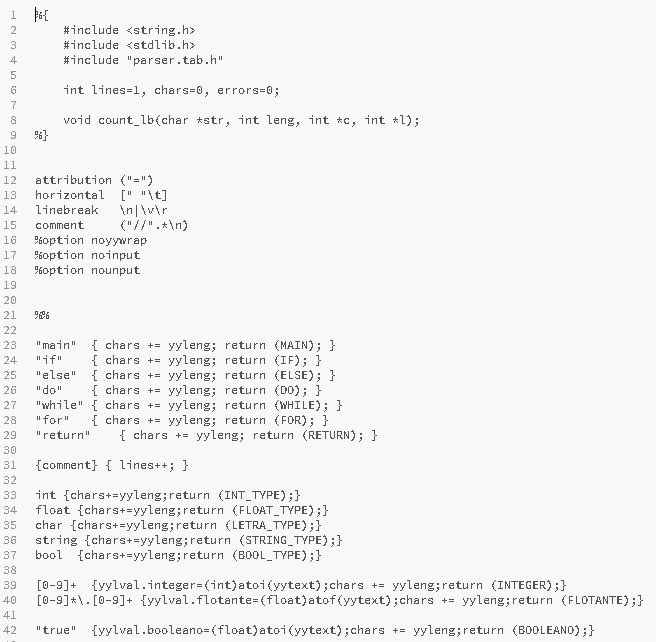
REGLAS GRAMATICALES

%%

CÓDIGO C ADICIONAL

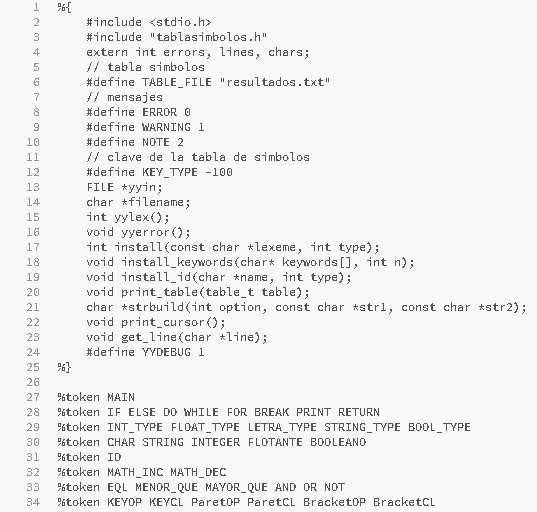
Siendo los ‘%%’, ‘% {‘.y ‘%}’, signos de puntuación utilizados para separar las secciones.

## **Código en Flex**



**Fig. 4** Código del Analizador Lexicográfico

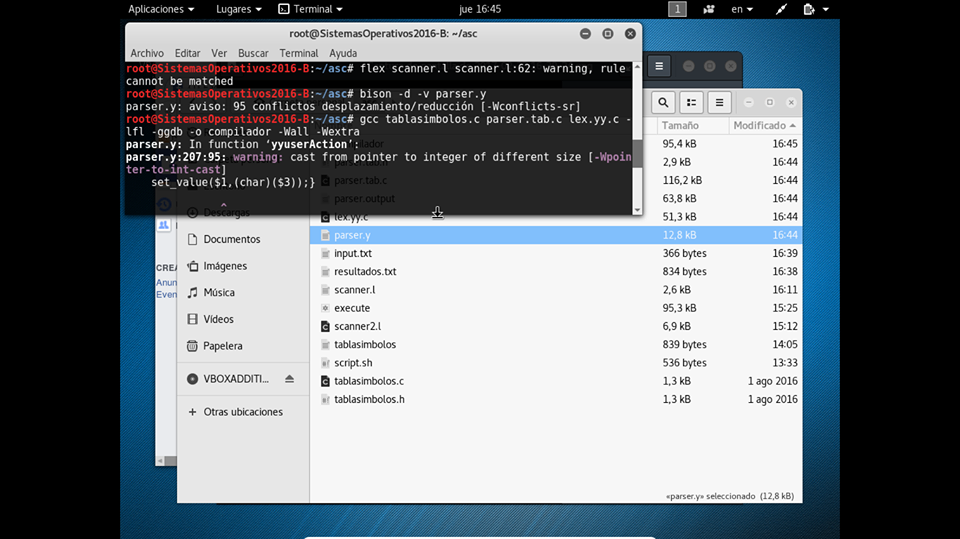
## **Código en Bison**



**Fig. 5** Código del Bison

# **Ejecución del proyecto**

## **Creación del ejecutable**



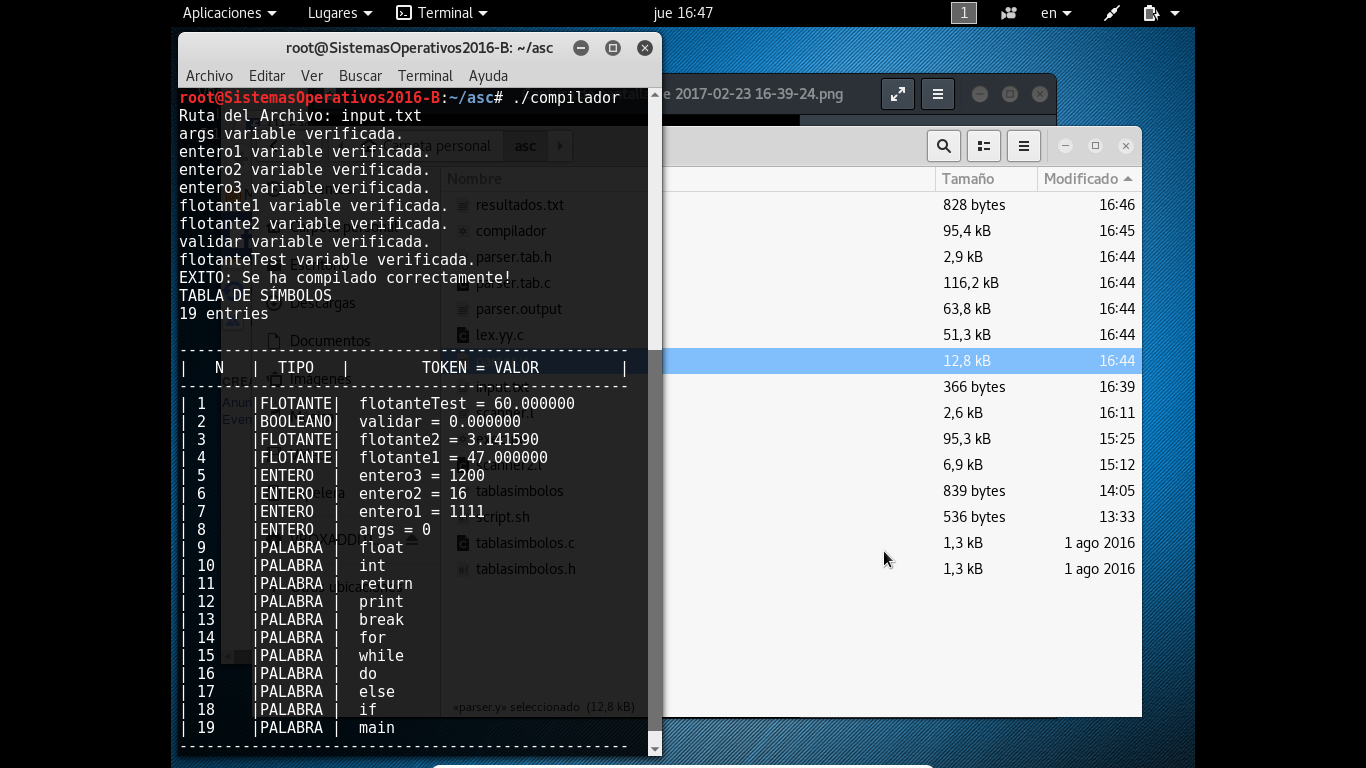
**Fig. 7** Compilación de los archivos “. l” y “. y “ y creación del ejecutable en la consola de comandos.

## **Archivo .txt a compilar**



**Fig. 8** Prueba de ejecución del Compilador

## **Resultados y Tabla de Símbolos**



**Fig. 9** Resultados del Compilador

## **Análisis de resultados**

Como se ve en la tabla de símbolos todos los identificadores son almacenados con su valor correspondiente, al igual que las palabras claves pertenecientes al texto ingresado por el usuario.

Los errores sintácticos correspondientes al texto ingresado son presentados correctamente con su número de línea de código correspondiente, también identificando el token correspondiente al error.

Después de la compilación del texto con el código se genera dos archivos la tabla de símbolos y el “parser.output” el cual será el que contendrá todo el detalle del proceso de la compilación.

# **Conclusión**

La compilación del texto se realizó correctamente, presentado la tabla de símbolos correspondiente y errores en el caso de comandos erróneos en el texto.

Las herramientas flex y bison resultaron de gran ayuda en la creación del analizador lexicográfico y el parser, la implementación de código se presenta en la **Fig. 4** y **Fig. 5.**

Para el ingreso del texto se debe tener en cuenta los parámetros definidos en el parser como son: iniciar con una **función main**, definir al menos una función dentro del **main**, las variables deben ser definidas por: **tipo:nombre**, etc.

**Referencias**

|  |  |
| --- | --- |
| [1] | S. G. R. María del Mar Aguilera Sierra, «lcc.uma.es - Traductores, Compiladores e Intérpretes,» [En línea]. Available: http://www.lcc.uma.es/~galvez/ftp/tci/tictema3.pdf. |
| [2] | S. G. R. María del Mar Aguilera Sierra, «lcc.uma.es - Traductores, Compiladores e Intérpretes "Tabla de Simbolos",» [En línea]. Available: http://www.lcc.uma.es/~galvez/ftp/tci/tictema5.pdf. |
| [3] | Ecured, «Lenguaje ensamblador,» [En línea]. Available: https://www.ecured.cu/Lenguaje\_ensamblador. [Último acceso: 22 Febrero 2017]. |

1. [↑](#footnote-ref-1)