



TECNOLÓGICO  
NACIONAL DE MÉXICO



INSTITUTO TECNOLÓGICO DE VERACRUZ

## DISEÑO DEL LENGUAJE

"REG"

### CARRERA:

Ingeniería en sistemas computacionales

### CATEDRÁTICA:

Ing. Ofelia Gutiérrez Giraldi.

### MATERIA:

Lenguajes y Autómatas I.

### HORA:

14:00-15:00 PM.



# ÍNDICE

Introducción .....	3
Justificación .....	4
Objetivo.....	4
Nombre del lenguaje.....	4
Especificaciones técnicas .....	5
El análisis léxico .....	9
Definición de variables.....	10
Ejemplo de código .....	10
Par ordenado.....	11
Análisis sintáctico.....	12
Manejo de errores .....	14
Tabla de errores léxicos .....	15
Tabla de errores sintácticos .....	15
Instrucciones de compilado sintáctico .....	16
Manual de usuario. ....	17

## Introducción

Un analizador léxico es un módulo destinado a leer caracteres del archivo de entrada, donde se encuentra la cadena a analizar, reconocer subcadenas que correspondan a símbolos del lenguaje y retornar los tokens correspondientes y sus atributos. Escribir analizadores léxicos eficientes “a mano” puede resultar una tarea tediosa y complicada, para evitarla se han creado herramientas de software – los generadores de analizadores léxicos – que generan automáticamente un analizador léxico a partir de una especificación provista por el usuario.

Todas estas herramientas para generar analizadores léxicos permiten definir la sintaxis de los símbolos mediante expresiones regulares, mientras que sus atributos deben ser computados luego del reconocimiento de una subcadena que constituya un símbolo del lenguaje. Una alternativa sería contar con una herramienta que permita computar los atributos a medida que se reconocen dichas subcadenas aprovechando el mecanismo de computo garantizado por el analizador léxico. De esta manera se libra al usuario del control sobre la cadena a computar.

## Justificación

Hoy en día es complicado, para un usuario o estudiante inexperto en el área de la programación, cambiar de un lenguaje a otro. Resulta muy difícil adaptarse de uno a otro.

Debido a este conflicto es que se pensó en crear 'REG', un lenguaje de programación basado en C++ y Java. reutilizaron las palabras reservadas más 'populares' de estos lenguajes y se combinaron en un solo lenguaje para que le sea más sencillo al usuario/estudiante adaptarse a los cambios de un lenguaje a otro.

## Objetivo

Utilizar las instrucciones de los lenguajes conocidos con el fin de adaptarse más fácil al cambiar a un nuevo lenguaje utilizando las palabras reservadas de los lenguajes c++ y java.

## Nombre del lenguaje

El nombre del lenguaje y del compilador se decidió que sería las iniciales de los nombres de los creadores de dichos programas que en este caso serían Rodrigo, Eduardo, Gustavo resultando como nombre REG

## Especificaciones técnicas

Sistema operativo: Windows 10

Lenguaje de programación: Java CC

Herramientas: Notepad++ o Bloc de notas

Versión de Java CC: 5.0

Jdk1.8.0\_181

Para que se pueda ejecutar el lenguaje se debe tener instalado el jdk y java cc en las variables de entorno del sistema de la siguiente manera:

Tanto para javacc como para el jdk

Paso 1: copiar la ruta de la ubicación del archivo bin dentro de javacc o el jdk

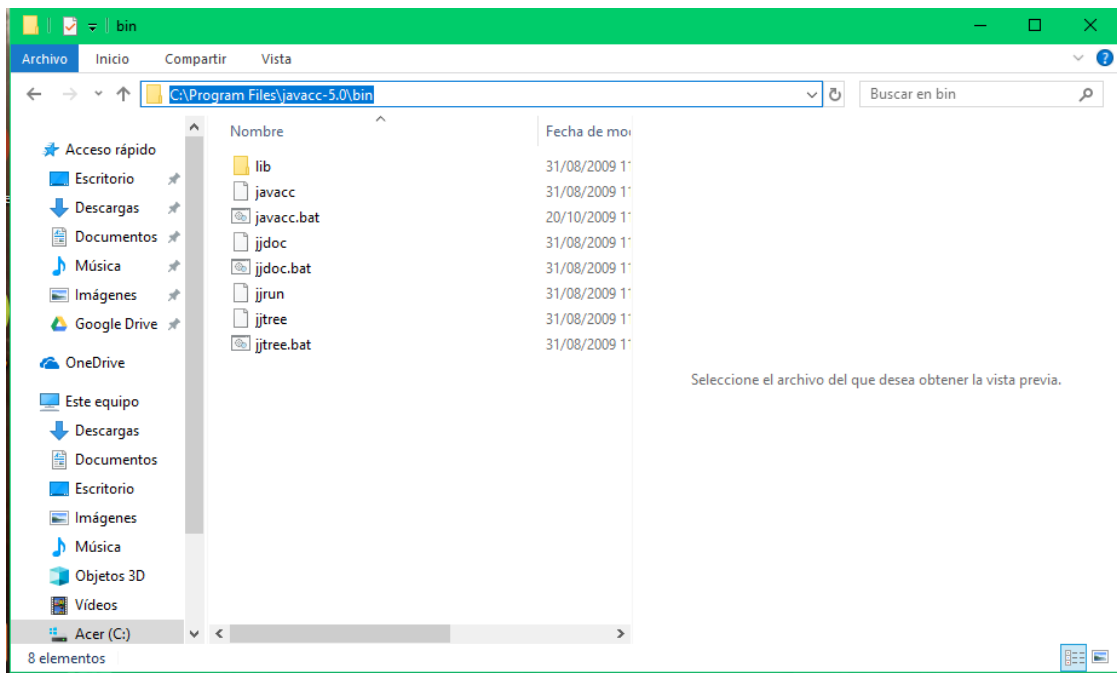


figura 1.- descripción del paso 1

## Paso 2: clic derecho en equipo y luego clic en propiedades

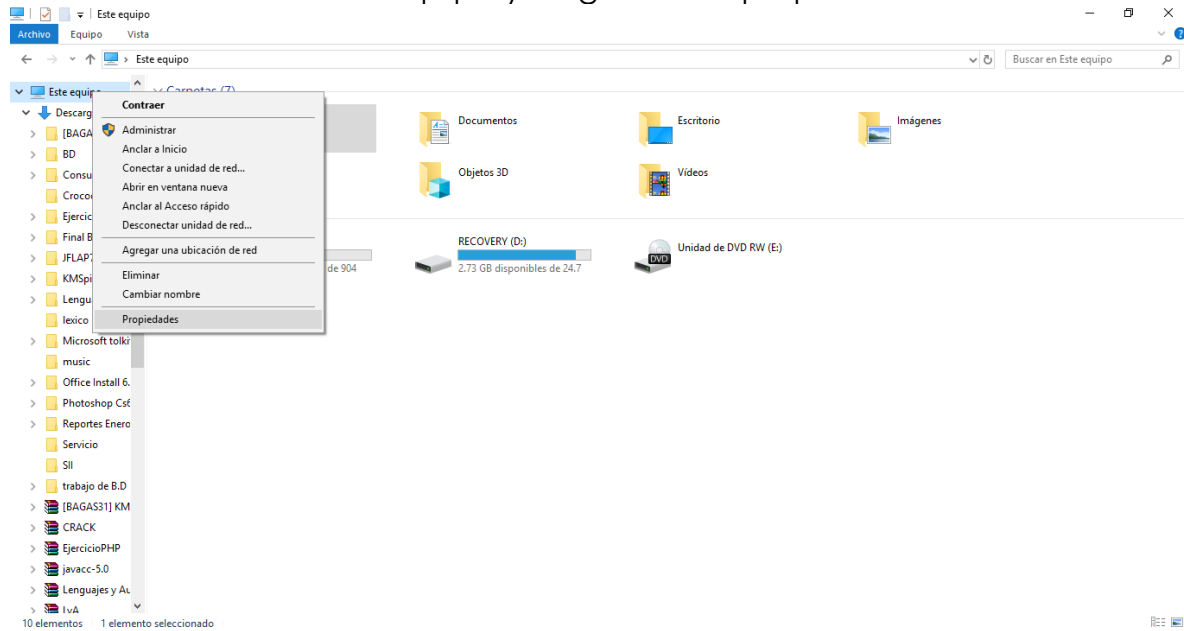


figura 2.- descripción grafica del paso 2

## Paso 3: clic derecho en configuración avanzada del sistema

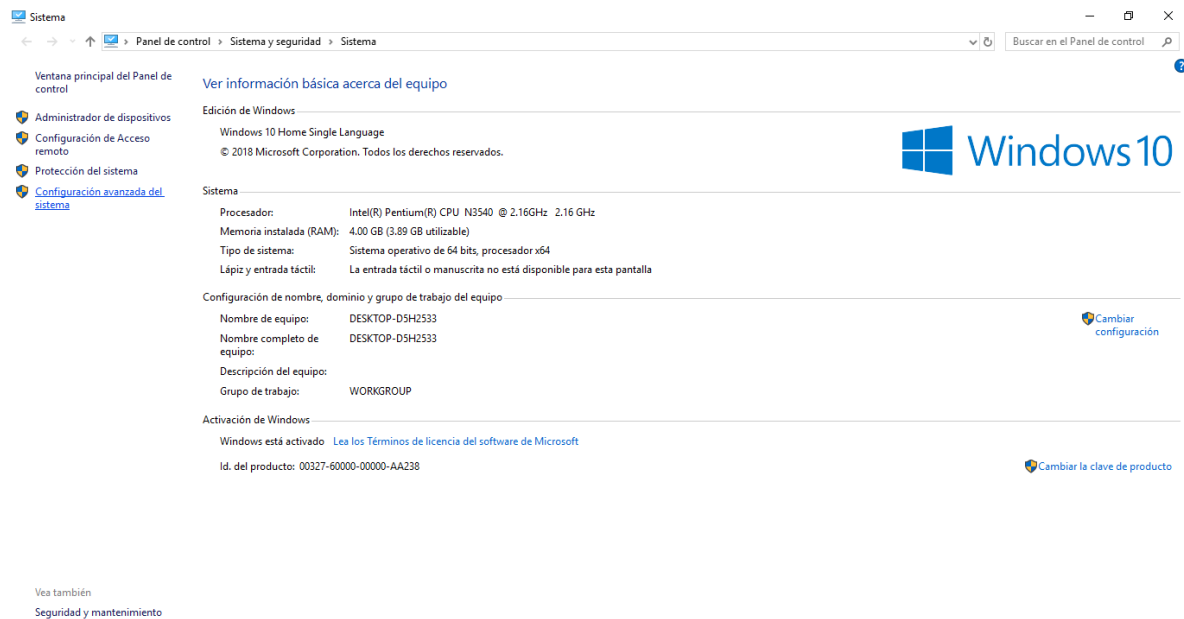


figura 3.- descripción grafica del paso 3

## Paso 4: clic en variables de entorno

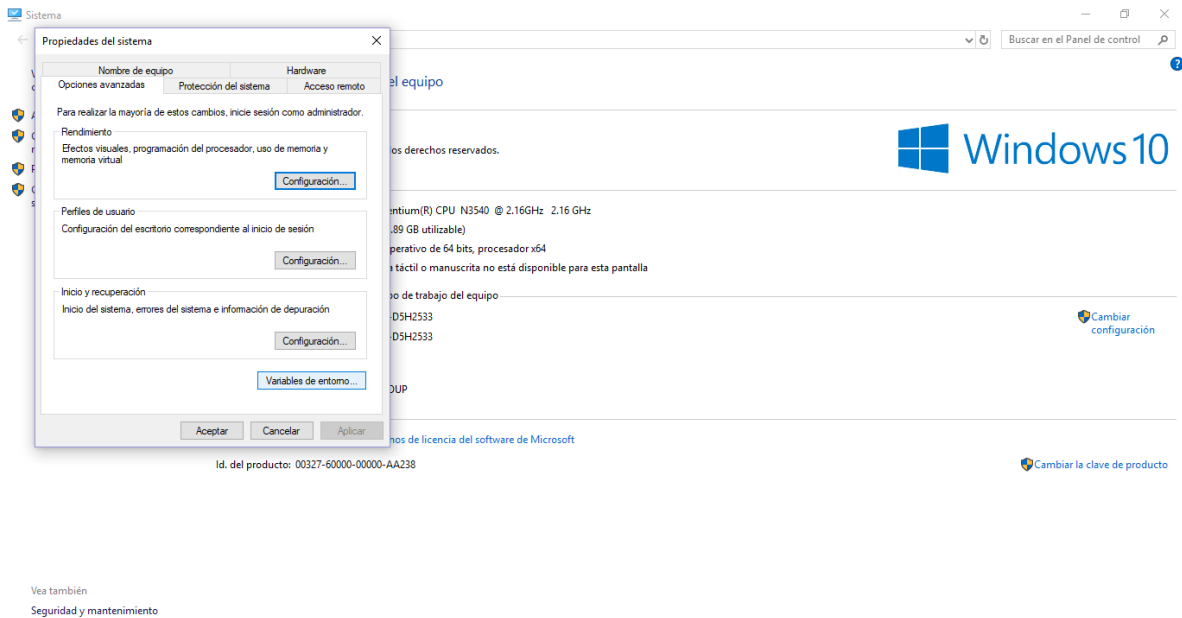


figura 4.- descripción grafica del paso 4

## Paso 5: clic en variable del sistema "path"

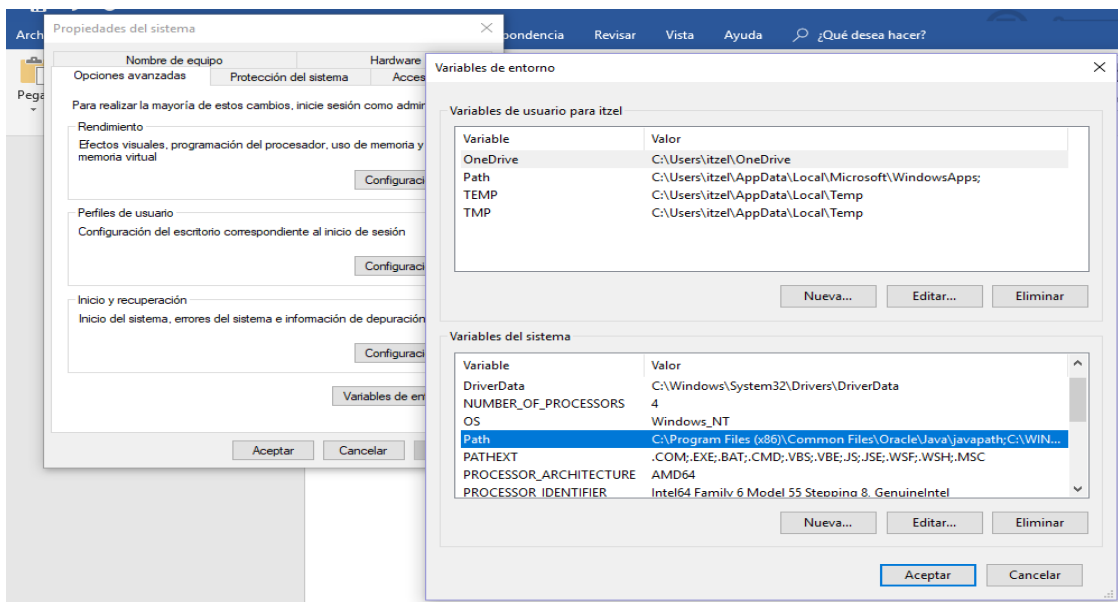


figura 5.- descripción grafica del paso 5

Paso 7: clic en nuevo y se pega la dirección del archivo inicialmente copiado (paso 1)

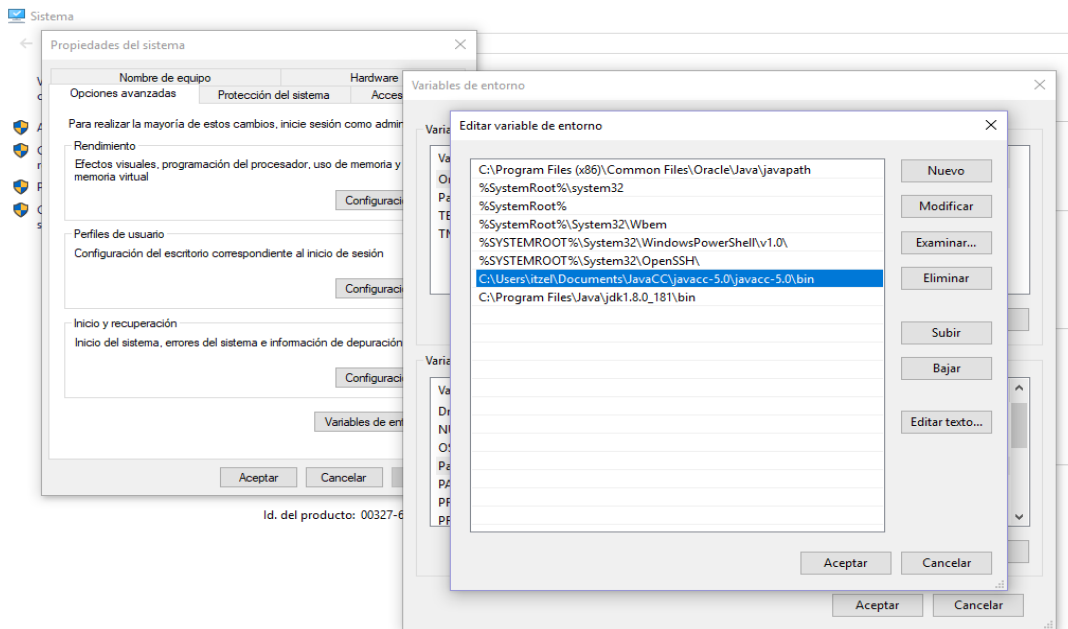


figura 6.- descripción grafica de los pasos 7



## El análisis léxico

En este tema vamos a proceder a definir y comprender las tareas que realiza el analizador léxico y que son clave para el correcto funcionamiento del compilador.

Como vemos en la figura, tiene como entrada el código fuente del lenguaje de programación que acepta el compilador y como salida proporciona al analizador sintáctico los tokens.

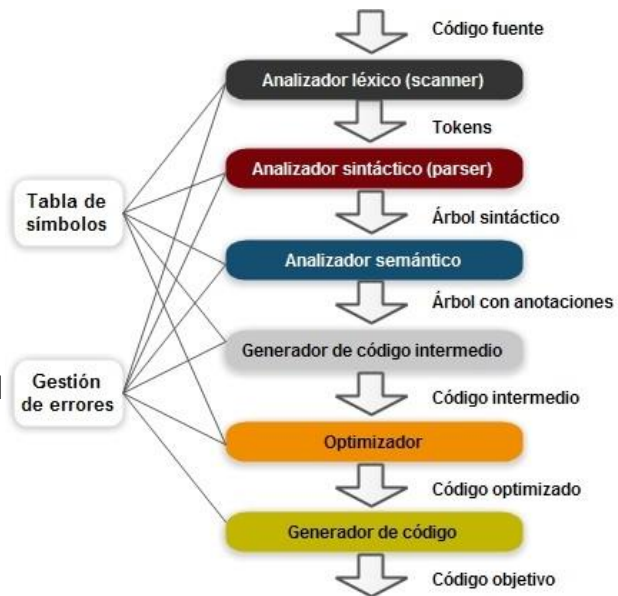


Figura. -7 fases de un compilador

si observamos con más detalle lo que pasa en el analizador léxico, en su relación con el analizador sintáctico y con la tabla de símbolos, vemos que una vez empieza a leer el código fuente y reconoce el primer token, se lo envía al analizador sintáctico y este, en cuanto lo recibe, le pide el siguiente token para que siga reconociendo la entrada. Por tanto, los tokens son enviados al analizador sintáctico **bajo demanda**.

Esta forma de funcionar se denomina "**dirigida por el analizador sintáctico**" (en inglés, *parser driven*).

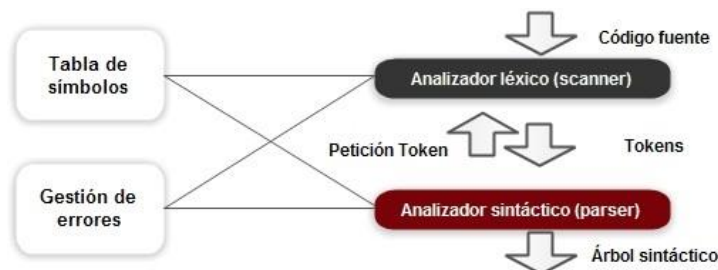


Figura 8.- función entre análisis léxico y sintáctico

Por otro lado, si reconoce un identificador lo almacena en la tabla de símbolos, y posteriormente, si el analizador sintáctico reconoce que ese identificador lleva asociada **información de tipo** (entero, real, etc.) o de valor, también añade esta información a la mencionada tabla.

En cuanto al sistema de gestión de errores, se encarga de **detectar símbolos que no pertenezcan a la gramática** porque no encajen con ningún patrón. Bien porque haya caracteres inválidos, ejemplo @, o bien porque se escriban mal las palabras reservadas del lenguaje, los identificadores o los números, como 5.25 en lugar de 5,25, pudiendo simplemente no hacer nada o bien informar del tipo de error que se ha cometido. Se puede **minimizar el número de errores** borrando caracteres inválidos, insertando el carácter que falta o remplazando un carácter por otro según sea el caso.

## Definición de variables

- Deben iniciar con un carácter \$
- Puede contener caracteres alfanuméricos
- Puede contener los siguientes símbolos: \_ siempre que no estén solos y no

inicien con ellos

Declaración de variables del lenguaje REG

```
"$"(["a"-"z", "A"-"Z"])(["a"-"z", "A"-"Z", "0"-"9", "_"])*
```

### Ejemplos:

```
$num1_  
$num
```

## Ejemplo de código

programa \$suma1

```
{ public static void Main ()  
    {
```

```

int $n1;

int $n2;

$N2 = $n3 + $n4;

    if ( $x == 0 ) {

        $x = $n4 + $n3;

    }

for ( int $x = 0; $x < 10; $x++ ) {

    string $name = " Hola Mundo! ";

}

}

}

```



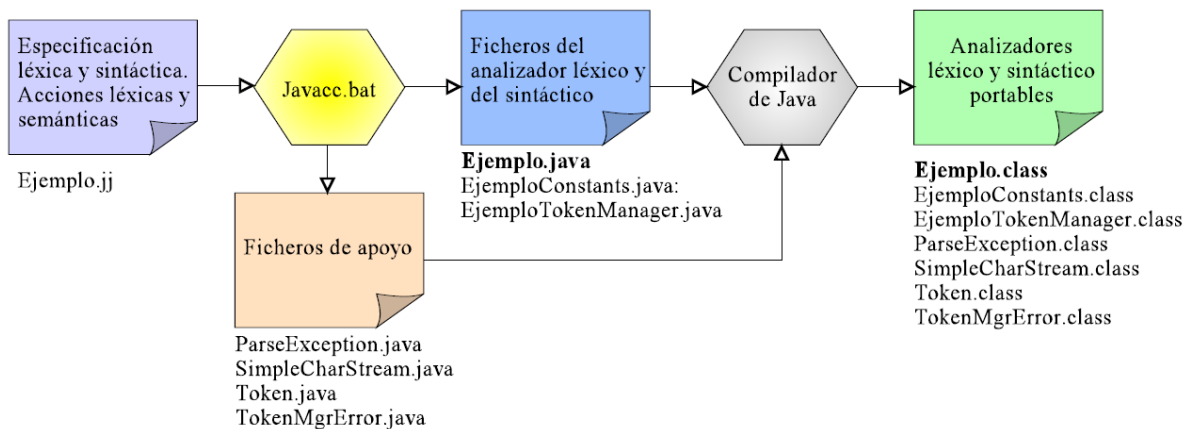
## Par ordenado

TIPO	VALOR
<b>programa</b>	PR_PROGRAMA
<b>public static void Main()</b>	MAIN
<b>scanner</b>	PALABRA_RESERVADA_ENTRADA
<b>println</b>	PALABRA_RESERVADA_SALIDA
<b>if</b>	PALABRA_RESERVADA_DE_CICLO_CONDICIONAL
<b>for</b>	PALABRA_RESERVADA_DE_CICLO_CONDICIONAL
<b>while</b>	PALABRA_RESERVADA_DE_BUBLE
<b>else</b>	PALABRA_RESERVADA
<b>do</b>	PALABRA_RESERVADA
<b>return</b>	PALABRA_RESERVADA
<b>switch</b>	PALABRA_RESERVADA_CONDICIONAL_SELECCION
<b>case</b>	PALABRA_RESERVADA_OPCION_SELECCION
<b>break</b>	PALABRA_RESERVADA
<b>;</b>	DELIMITADOR_PUNTO_COMA
<b>{</b>	DELIMITADOR_LLAVE_ABRE
<b>}</b>	DELIMITADOR_LLAVE_CIERRA
<b>(</b>	DELIMITADOR_PARENTESIS_ABRE
<b>)</b>	DELIMITADOR_PARENTESIS_CIERRA
<b>,</b>	DELIMITADOR_COMA

:	DELIMITADOR_DOSPUNTOS
+	OPERADOR_ARITMETICO
-	OPERADOR_ARITMETICO
*	OPERADOR_ARITMETICO
/	OPERADOR_ARITMETICO
%	OPERADOR_MOD
=	OPERADOR_ASIG
^	OPERADOR_POT
<	OPERADOR_MENOR
>	OPERADOR_MAYOR
<=	OPERADOR_MENOR_IGUAL
>=	OPERADOR_MAYOR_IGUAL
==	OPERADOR_IGUAL
!=	OPERADOR_DIF
&&	OPERADOR_LOGICO
	OPERADOR_LOGICO
!	OPERADOR_LOGICO
++	OPERADOR_INCREMENTO
--	OPERADOR_DECREMENTO
string	DATO_TIPO_STRING
int	DATO_TIPO_INT
float	DATO_TIPO_FLOAT
boolean	DATO_TIPO_BOOLEAN

## Análisis sintáctico

Es la fase del analizador que se encarga de chequear la secuencia de tokens que representa al texto de entrada, en base a una gramática dada. En caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce en base a una representación computacional. Este árbol es el punto de partida de la fase posterior de la etapa de análisis: el analizador semántico



**Figura 9** - Posición del analizador sintáctico en el modelo del compilador

Pero esto es la teoría; en la práctica, el analizador sintáctico dirige el proceso de compilación, de manera que el resto de las fases evolucionan a medida que el sintáctico va reconociendo la secuencia de entrada por lo que, a menudo, el árbol ni siquiera se genera realmente.

En la práctica, el analizador sintáctico también:

- Incorpora acciones semánticas en las que coloca en el resto de fases del compilador (excepto el analizador léxico): desde el análisis semántico hasta la generación de código.
- Informa de la naturaleza de los errores sintácticos que encuentra e intenta recuperarse de ellos para continuar la compilación.
- Controla el flujo de tokens reconocidos por parte del analizador léxico.

En definitiva, realiza casi todas las operaciones de la compilación, dando lugar a un método de trabajo denominado compilación dirigida por sintaxis.

## Manejo de errores

Si un compilador tuviera que procesar sólo programas correctos, su diseño e implementación se simplificarían mucho. Las primeras versiones de los programas suelen ser incorrectas, y un buen compilador debería ayudar al programador a identificar y localizar errores. Es más, considerar desde el principio el manejo de errores puede simplificar la estructura de un compilador y mejorar su respuesta a los errores.

Los errores en la programación pueden ser de los siguientes tipos:

- **Léxicos**, producidos al escribir mal un identificador, una palabra clave o un operador.
- **Sintácticos**, por una expresión aritmética o paréntesis no equilibrados.
- **Semánticos**, como un operador aplicado a un operando incompatible.
- **Lógicos**, puede ser una llamada infinitamente recursiva.
- **De corrección**, cuando el programa no hace lo que el programador realmente deseaba.

El manejo de errores de sintaxis es el más complicado desde el punto de vista de la creación de compiladores. Nos interesa que cuando el compilador encuentre un error, no cancele definitivamente la compilación, sino que se recupere y siga buscando errores. Recuperar un error no quiere decir corregirlo, sino ser capaz de seguir construyendo el árbol sintáctico a pesar de los errores encontrados. En vista de esto, el manejador de errores de un analizador sintáctico debe tener como objetivos:

- **Indicar los errores de forma clara y precisa.** Debe informar mediante los correspondientes mensajes del tipo de error y su localización.
- **Recuperarse del error**, para poder seguir examinando la entrada.
- **Distinción entre errores y advertencias.** Las advertencias se suelen utilizar para informar sobre sentencias válidas pero que, por ser poco frecuentes, pueden constituir una fuente de errores lógicos.
- **No ralentizar** significativamente la compilación.

## Tabla de errores léxicos

Error léxico	Ejemplos de errores
Utilizar caracteres que no pertenecen al lenguaje	#, &,  , °, ¬, \, ", ~, _ , ~
Usar una cadena que no coincida con los patrones de lenguaje	Num#a12 o \$num-23u
Errores de ortografía	Inc, íf, públic

## Tabla de errores sintácticos

Error sintáctico	Ejemplo de errores
Falta del delimitador"; "en una sentencia	String \$letra o \$num + \$num
Falta del delimitador "("o")" en la estructura del código	If{\$x<=\$num{ }
Falta del delimitador "{"o"}"	If{\$x<=\$num){
No corresponde a la estructura del programa	If[\$num<\$num1]{ }

## Instrucciones de compilado sintáctico

1. Abrimos el CMD
2. Tecleamos comando cd y la ruta donde esta guardado el archivo con la extensión **.jj**
3. Compilamos el archivo con el siguiente comando **javacc REG.jj**
4. Compilamos todo con la extensión .java con el siguiente comando:  
**javac \*.java**
5. Despues para ejecutar un ejemplo se ocupa la siguiente instrucción:  
**java REG** < con el nombre del archivo y su extensión



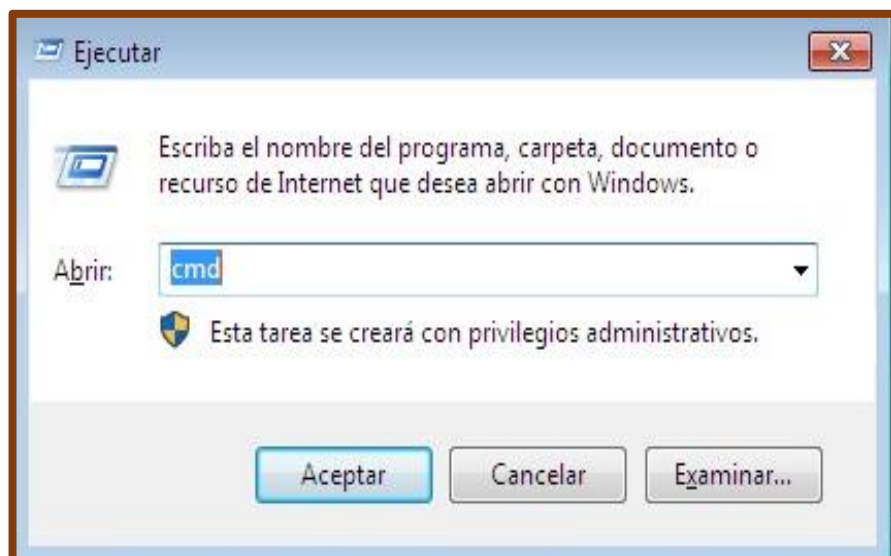
# Manual de usuario.



1.-Primero accedemos al cmd.

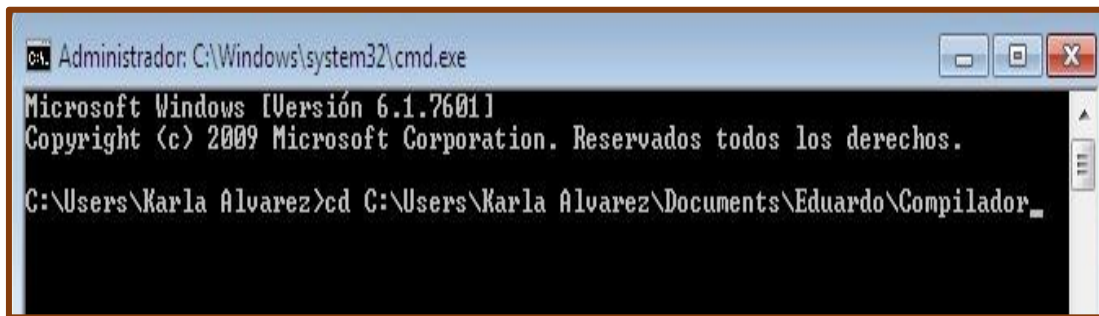
Apretamos la tecla Windows seguido con la tecla R

Nos abrirá una pestaña donde tecleamos cmd para acceder al Simbolo del Sistema.



**Figura 1.- Acceder al CMD**

2.- Estando ahí ingresaremos cd y la ruta donde se encuentra nuestro compilador.

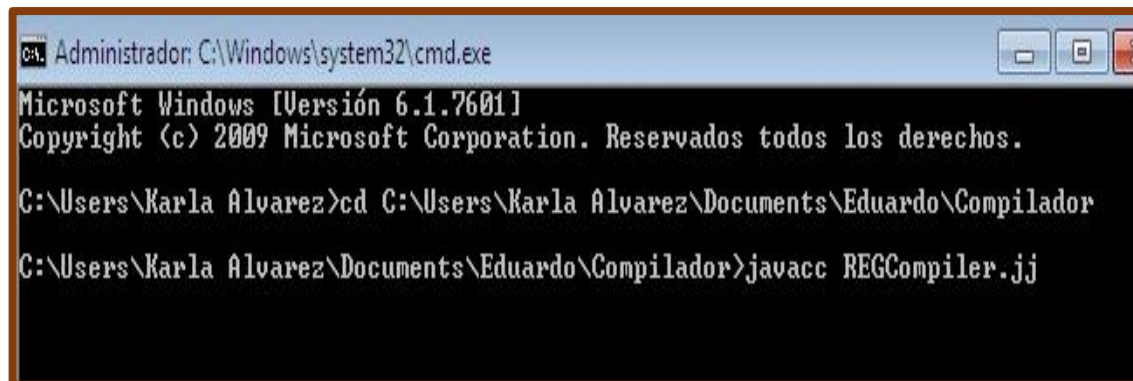


```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Karla Alvarez>cd C:\Users\Karla Alvarez\Documents\Eduardo\Compilador_
```

**Figura 1.1.- Accerder a la Ruta del Compilador**

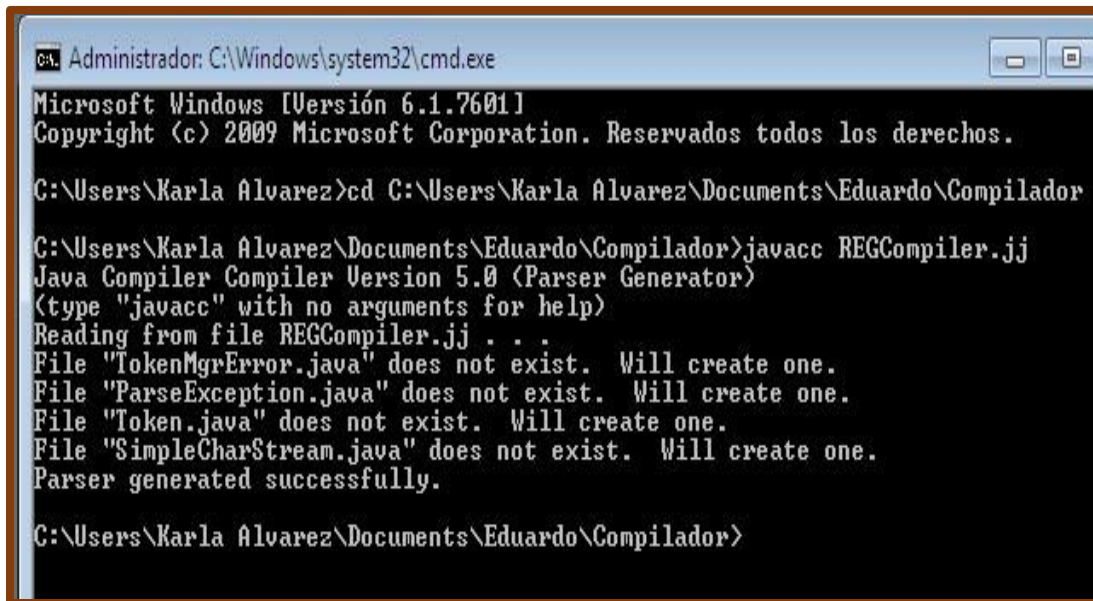
3.- Agregamos javacc REG.jj Lo que esto hara es que metacompilara el archivo.



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Karla Alvarez>cd C:\Users\Karla Alvarez\Documents\Eduardo\Compilador
C:\Users\Karla Alvarez\Documents\Eduardo\Compilador>javacc REGCompiler.jj
```

**Figura 1.2.- Metacompilamos el Archivo REGCompiler.jj**



```
CA. Administrador: C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

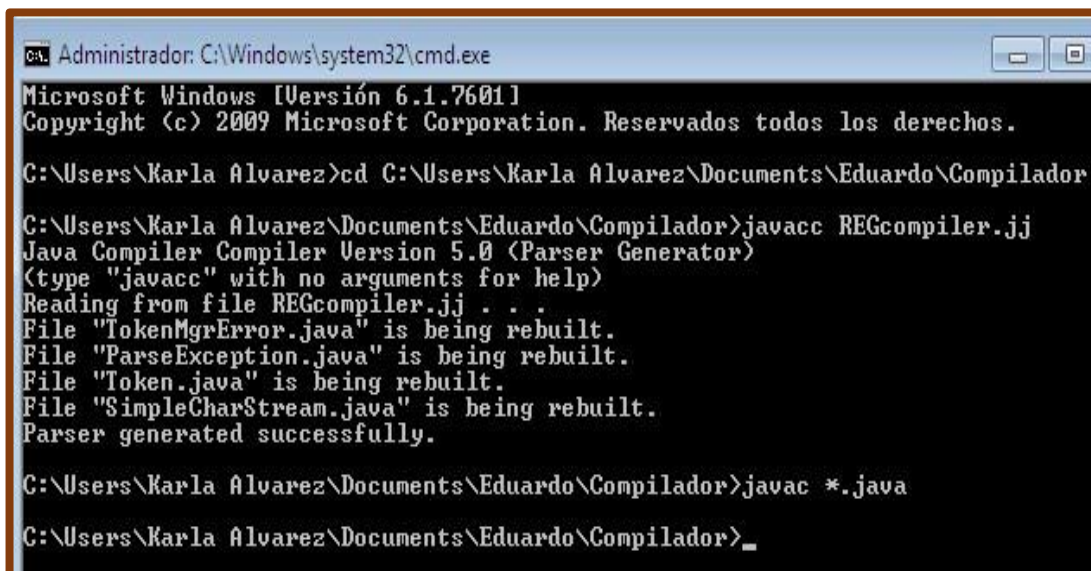
C:\Users\Karla Alvarez>cd C:\Users\Karla Alvarez\Documents\Eduardo\Compilador

C:\Users\Karla Alvarez\Documents\Eduardo\Compilador>javacc REGCompiler.jj
Java Compiler Compiler Version 5.0 (Parser Generator)
<type "javacc" with no arguments for help>
Reading from file REGCompiler.jj . . .
File "TokenMgrError.java" does not exist. Will create one.
File "ParseException.java" does not exist. Will create one.
File "Token.java" does not exist. Will create one.
File "SimpleCharStream.java" does not exist. Will create one.
Parser generated successfully.

C:\Users\Karla Alvarez\Documents\Eduardo\Compilador>
```

**Figura 1.3.- Resultado de la Metacompilación del Archivo REGCompiler.jj**

4.- Luego compilar todos los archivos .java



```
CA. Administrador: C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Karla Alvarez>cd C:\Users\Karla Alvarez\Documents\Eduardo\Compilador

C:\Users\Karla Alvarez\Documents\Eduardo\Compilador>javacc REGcompiler.jj
Java Compiler Compiler Version 5.0 (Parser Generator)
<type "javacc" with no arguments for help>
Reading from file REGcompiler.jj . . .
File "TokenMgrError.java" is being rebuilt.
File "ParseException.java" is being rebuilt.
File "Token.java" is being rebuilt.
File "SimpleCharStream.java" is being rebuilt.
Parser generated successfully.

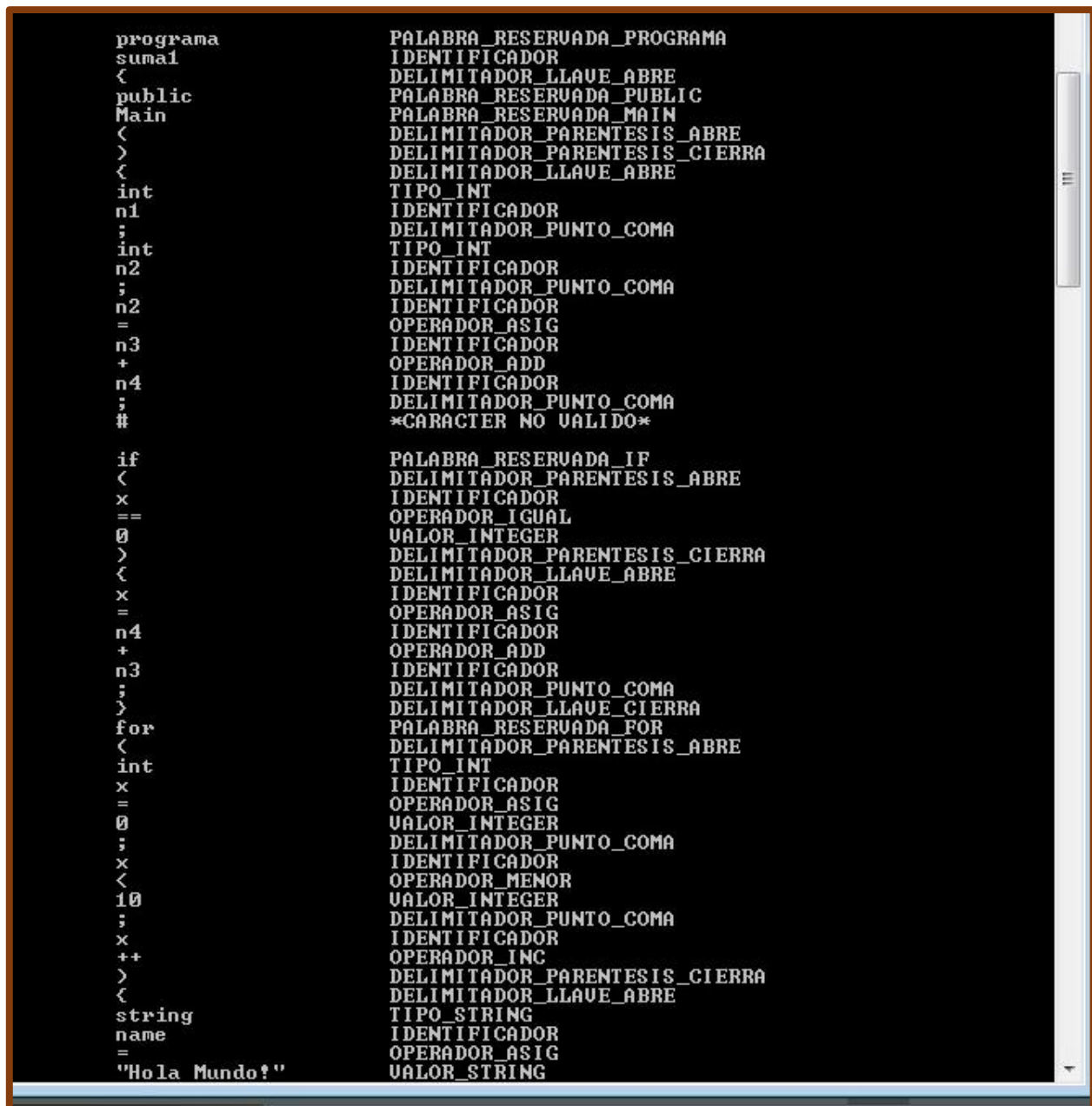
C:\Users\Karla Alvarez\Documents\Eduardo\Compilador>javac *.java

C:\Users\Karla Alvarez\Documents\Eduardo\Compilador>_
```

**Figura 1.4.- Compilación de todos los archivos .java**

5.-Podemos ejecutar el analizador de 2 maneras.

A) java REG Ejemplo código.txt

The image shows a screenshot of a Java application window titled 'REG'. The window contains two columns of text. The left column is a Java code snippet, and the right column shows the corresponding lexical tokens identified by the program. The tokens are listed in all caps and include reserved words, identifiers, operators, and delimiters. The code snippet is as follows:

```
programa
suma1
{
public
Main
{
int
n1
;
int
n2
;
n2
=
n3
+
n4
;
#

if
{
x
==
0
}
{
x
=
n4
+
n3
;
}
for
{
int
x
=
0
;
x
<
10
;
x
++
}
string
name
=
"Hola Mundo!"
```

The corresponding tokens are:

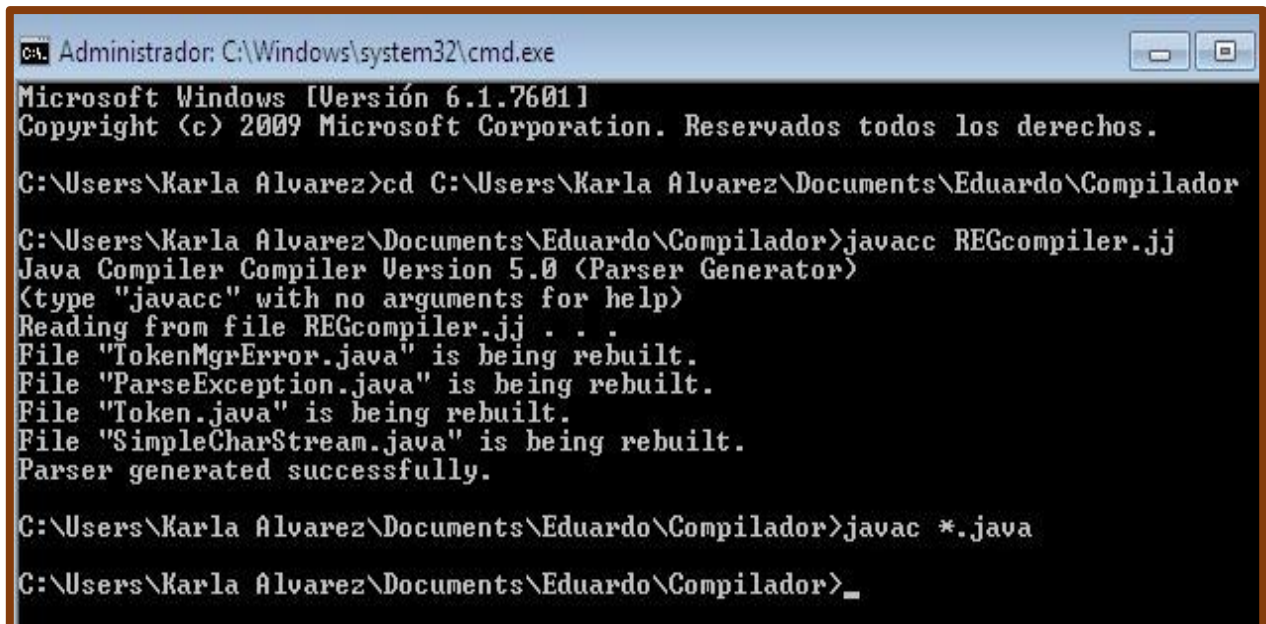
```
PALABRA_RESERVADA_PROGRAMA
IDENTIFICADOR
DELIMITADOR_LLAUE_ABRE
PALABRA_RESERVADA_PUBLIC
PALABRA_RESERVADA_MAIN
DELIMITADOR_PARENTESIS_ABRE
DELIMITADOR_PARENTESIS_CIERRA
DELIMITADOR_LLAUE_ABRE
TIPO_INT
IDENTIFICADOR
DELIMITADOR_PUNTO_COMA
TIPO_INT
IDENTIFICADOR
DELIMITADOR_PUNTO_COMA
IDENTIFICADOR
OPERADOR_ASIG
IDENTIFICADOR
OPERADOR_ADD
IDENTIFICADOR
DELIMITADOR_PUNTO_COMA
*CARACTER NO VALIDO*

PALABRA_RESERVADA_IF
DELIMITADOR_PARENTESIS_ABRE
IDENTIFICADOR
OPERADOR_IGUAL
VALOR_INTEGER
DELIMITADOR_PARENTESIS_CIERRA
DELIMITADOR_LLAUE_ABRE
IDENTIFICADOR
OPERADOR_ASIG
IDENTIFICADOR
OPERADOR_ADD
IDENTIFICADOR
DELIMITADOR_PUNTO_COMA
DELIMITADOR_LLAUE_CIERRA
PALABRA_RESERVADA_FOR
DELIMITADOR_PARENTESIS_ABRE
TIPO_INT
IDENTIFICADOR
OPERADOR_ASIG
VALOR_INTEGER
DELIMITADOR_PUNTO_COMA
IDENTIFICADOR
OPERADOR_MENOR
VALOR_INTEGER
DELIMITADOR_PUNTO_COMA
IDENTIFICADOR
OPERADOR_INC
DELIMITADOR_PARENTESIS_CIERRA
DELIMITADOR_LLAUE_ABRE
TIPO_STRING
IDENTIFICADOR
OPERADOR_ASIG
VALOR_STRING
```

Figura 1.5.- Resultado del Análisis Léxico.

B) java REG [ENTER]

6.- Para acabar el análisis tecleamos Ctrl C.



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Karla Alvarez>cd C:\Users\Karla Alvarez\Documents\Eduardo\Compilador

C:\Users\Karla Alvarez\Documents\Eduardo\Compilador>javacc REGcompiler.jj
Java Compiler Compiler Version 5.0 (Parser Generator)
<type "javacc" with no arguments for help>
Reading from file REGcompiler.jj . . .
File "TokenMgrError.java" is being rebuilt.
File "ParseException.java" is being rebuilt.
File "Token.java" is being rebuilt.
File "SimpleCharStream.java" is being rebuilt.
Parser generated successfully.

C:\Users\Karla Alvarez\Documents\Eduardo\Compilador>javac *.java

C:\Users\Karla Alvarez\Documents\Eduardo\Compilador>_
```

**Figura 1.6.- Utilizando Ctrl C (Terminar el Análisis Léxico)**