2013

**INFORME DE DISEÑO DE COMPILADORES**

**Primer parte**

**GRUPO N 13**

**INTEGRANTES:**

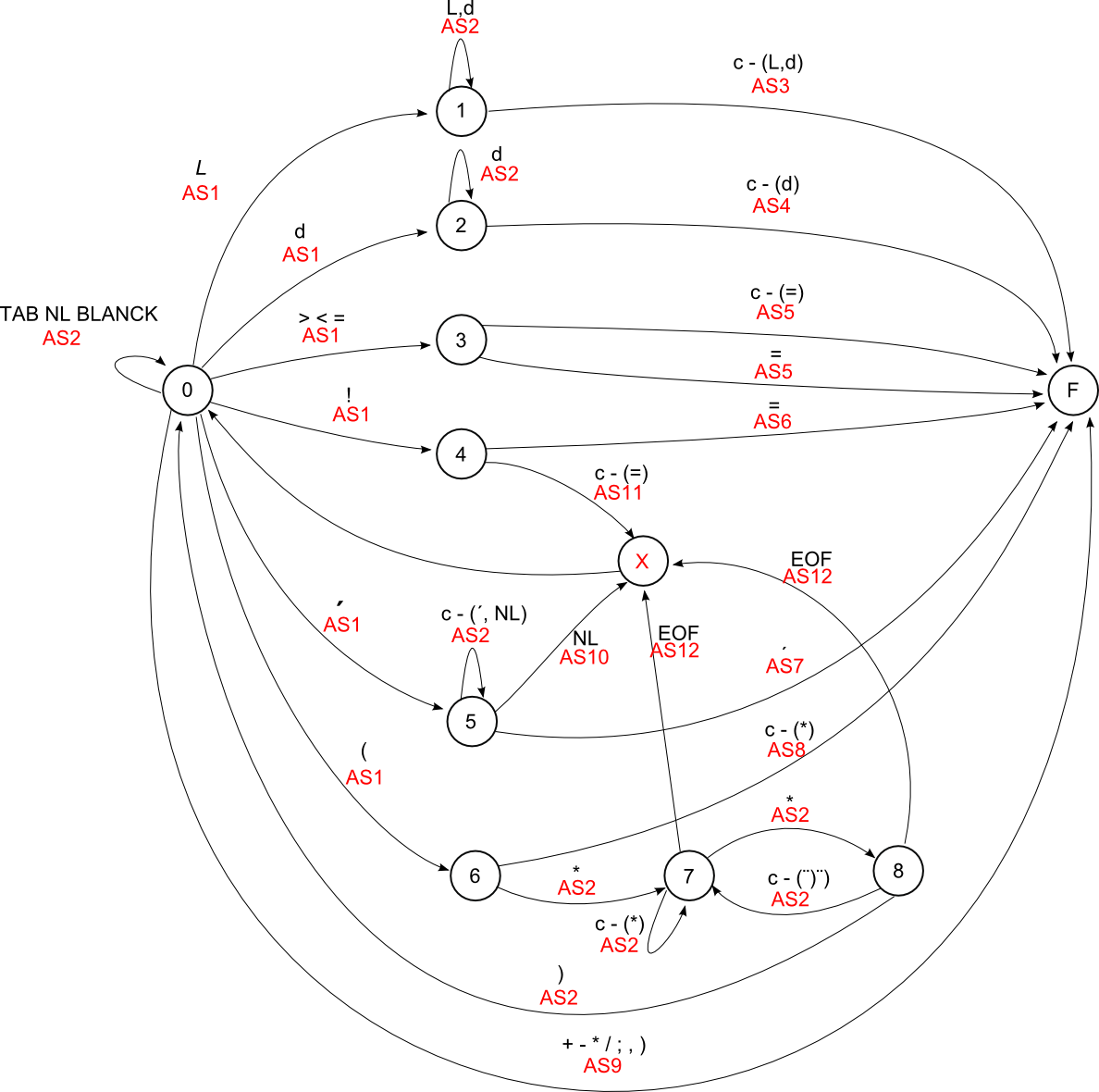
* **Fernadez, Sebastian (**[**sebachk@gmail.com**](mailto:sebachk@gmail.com)**)**
* **Gomez Ortiz, Ibrian (**[**ib.gomezo@gmail.com**](mailto:ib.gomezo@gmail.com)**)**
* **Pagano, Diego (**[**diegoipagano@gmail.com**](mailto:diegoipagano@gmail.com)**)**

El siguiente describirá y explicara la creación de analizador léxico como parte de un trabajo practico propuesto por la catedra de Diseño de Compiladores.  
  
**¿Qué es un analizador léxico?**   
Un analizador léxico (AL) es un componente de un compilador de lenguaje de programación, que se encarga de analizar y encontrar errores léxicos en un código fuente dado. El AL buscará palabras reservadas propias del lenguaje, además de detectar identificadores, constantes, ---------------. A estos elementos se los denominan *Tokens.*

**¿Cómo funciona?**  
El AL funciona en conjunto con otro componente llamado analizador sintáctico (AS). En éste caso el sintáctico se desarrollada utilizando una herramienta ya implementada que se denomina YACC. El AS es quien pone en funcionamiento al procesador léxico solicitándole un token.  
El AL trabaja recorriendo un archivo de texto caracter a caracter procesándolos en un autómata finito. Cada vez que se arribe a un estado final el analizador habrá encontrado un token que deberá ser entregado al procesador sintáctico.  
Es importante aclarar que el AL se pone en funcionamiento solo cada vez que el AS le solicita un token y que una vez encontrado y entregado, se detiene hasta nuevo pedido.  
Cada token está compuesto por un número que lo identifica y un valor de posición que indica en que parte de la tabla de símbolos está almacenado.

**Desarrollo de la aplicación**Como primer medida el grupo comenzó por la creación del autómata que describe los estados por los que atraviesa el AL cada vez que lee un carácter y hasta encontrar un token. (Imagen 1)  
  
L = carácter letra  
d = carácter digito   
TAB = carácter de tabulación  
NL = carácter de Nueva Línea  
BLANK = carácter de espacio  
c – () = cualquier carácter menos los que se encuentran especificados entre paréntesis  
Cualquier otro símbolo o carácter se representa así mismo.

Los escritos en rojos hacen referencia a las Acciones Semánticas que se ejecutan en cada transición de estados.



*Imagen 1 correspondiente al autómata que representa el Analizador Léxico*

Matriz de transición de estados

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Carácter** | | | | | | | | | | | | | | | | | |
| **Estado** | L | d | > | < | = | ! | ‘ | ( | ) | + | - | \* | / | ; | , | NL | TAB | BLANCK |
| 0 | 1 | 2 | 3 | 3 | 3 | 4 | 5 | 6 | F | F | F | F | F | F | F | 0 | 0 | 0 |
| 1 | 1 | 1 | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| 2 | F | 2 | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| 3 | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| 4 | **X** | **X** | **X** | **X** | F | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | F | **X** | **X** |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 | F | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | **X** | 5 | 5 |
| 6 | F | F | F | F | F | F | F | F | F | F | F | 7 | F | F | F | F | F | F |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 8 | 7 | 7 | 7 | 7 | 7 | 7 |
| 8 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 0 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| F |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

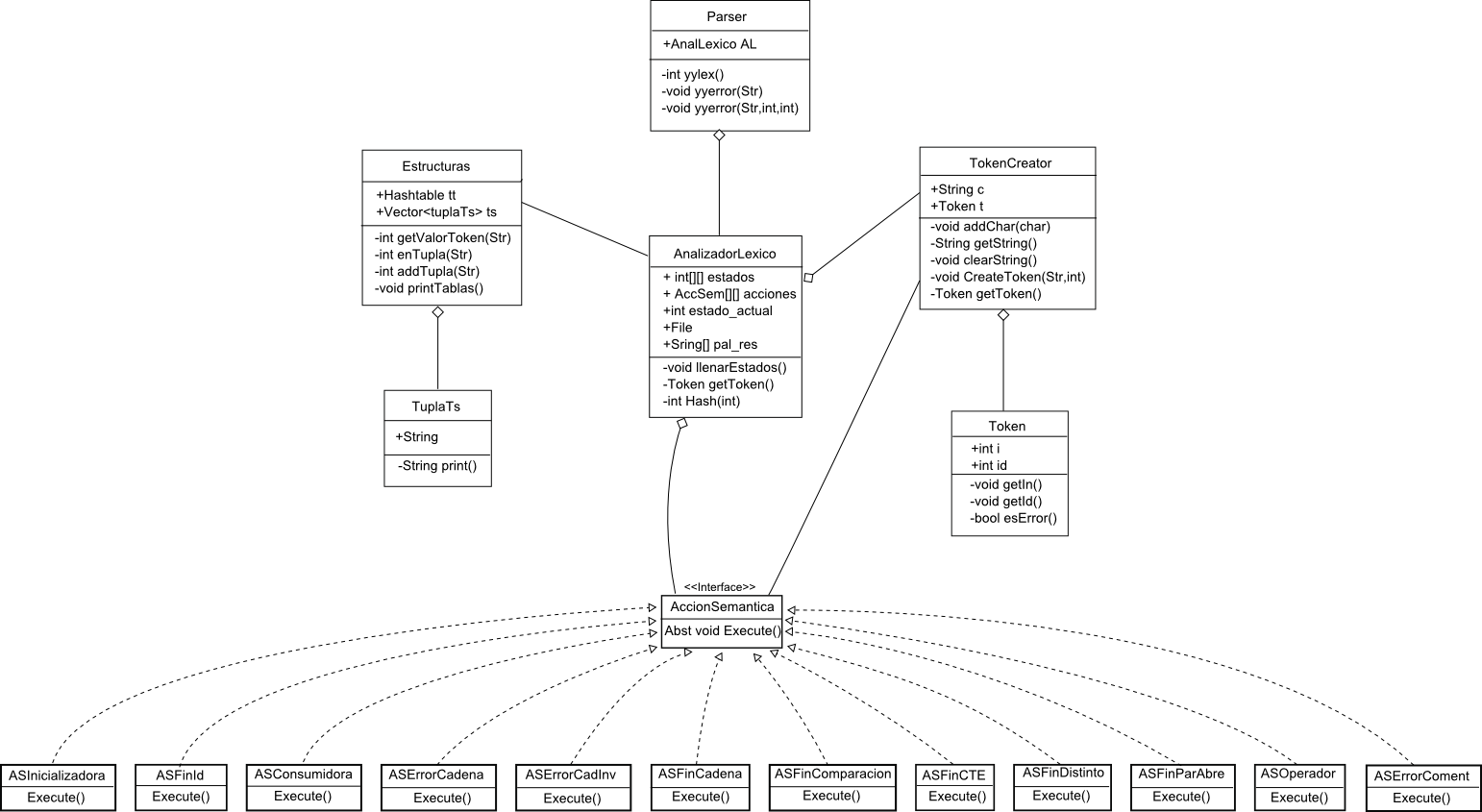
Los estados marcados con **X** son estados de error.

Matriz de Acciones semánticas

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Est** | | **Carácter** | | | | | | | | | | | | | | | | | |
|  | L | | d | > | < | = | ! | ‘ | ( | ) | + | - | \* | / | ; | , | NL | TAB | BLK |
| 0 | AS1 | | AS1 | AS1 | AS1 | AS1 | AS1 | AS1 | AS1 | AS9 | AS9 | AS9 | AS9 | AS9 | AS9 | AS9 | AS2 | AS2 | AS2 |
| 1 | AS2 | | AS2 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 | AS3 |
| 2 | AS4 | | AS2 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 | AS4 |
| 3 | AS5 | | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 | AS5 |
| 4 | AS11 | | AS11 | AS11 | AS11 | AS6 | AS11 | AS11 | AS11 | AS11 | AS11 | AS11 | AS11 | AS11 | AS11 | AS11 | AS11 | AS11 | AS11 |
| 5 | AS2 | | AS2 | AS2 | AS2 | AS2 | AS2 | AS7 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS10 | AS2 | AS2 |
| 6 | AS8 | | AS8 | AS8 | AS8 | AS8 | AS8 | AS8 | AS8 | AS8 | AS8 | AS8 | AS2 | AS8 | AS8 | AS8 | AS8 | AS8 | AS8 |
| 7 | AS2 | | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 |
| 8 | AS2 | | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 | AS2 |
| F |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Una vez delineado el autómata y las matrices que lo representan se comenzó con la implementación. El grupo eligió el lenguaje Java para desarrollarse.  
Se crearon las clases:

* Analizador Léxico: Encargada de recorrer un archivo de texto (código fuente) carácter a carácter y encontrar tokens y devolverlos. Una vez encontrado algún token se detiene hasta nuevo pedido.
* Estructuras: Contiene la tabla de símbolos (implementada como un vector<TuplaTs>) y la tabla de tokens (implementada con una hashtable). Contiene funcionalidades para agregar y consultar entradas en ambas tablas.
* TuplaTs: Implementa la tupla de la tabla de símbolos. Los atributos de ésta pueden variar
* TokenCreator: Se encarga de crear los tokens. Almacena los caracteres que se le van pasando y cuando se le solicita crea un token con los caracteres que venía almacenando
* Token: Clase que representa un token. Está compuesta de un entero que representa el numero de token y un puntero a la posición de la tabla de símbolos donde se encuentra almacenado.
* Interface Accion Semantica: Contiene el comportamiento común de toda acción Semantica.
* ASInicializadora: Lee el primer carácter, marca el archivo en la posición donde lo leyó y lo almacena
* ASConsumidora: Consume un carácter y lo almacena.
* ASFinId: Detecta un carácter que no pertenece a la cadena, no lo consume y vuelve hacia atrás una posición en el archivo. Crea un token y además trunca el tamaño del token a 15 caracteres.



* ASFinCTE: Detecta un carácter que no pertenece a la constante, no lo consume y vuelve hacia atrás una posición en el archivo. Crea un token y controla que el valor de la constante no supere el límite permitido
* ASFinComparacion: Detecta si el/los caracter/es de comparación está/n bien escrito/s. Se ejecuta siempre después de haber leído un > < ó =. Luego si el carácter que viene es un =, puede saber si se trata de un >=, <=, ==. Caso contrario vuelve una posición en el archivo.
* ASFinDistinto: Detecta el token != (distinto). Si lo encuentra lo crea.
* ASFinCadena: Detecta la comilla simple final y de esta manera reconoce un fin de cadena. La consume y crea el token correspondiente.
* ASFinParAbre: Si luego de leer un paréntesis que abre viene cualquier otro carácter que no es un asterisco estaremos entonces en el inicio de un nuevo token. Esta acción semántica retrocede una posición en la lectura del archivo y crea un token con el carácter “(”.
* ASOperador: Ésta acción detecta los operadores + - \* / ; , ) y crea el token correspondiente.
* ASErrorCadena: Detecta e informa el error provocado por cadenas que contienen en su interior el carácter de nueva línea.
* ASErrorCarInv: Detecta inversiones de caracteres. Se ejecuta luego de leer un ! cuando el analizador léxico espera un = y esto no ocurre.

La clase Parse y ParseVal surgen de ejecutar la aplicación YACC que recibe como parámetro la gramatica del lenguaje creado por el grupo. La gramatica se encuentra en el archivo Gramatica.y que se adjunta en ésta entrega.

**Problemas surgidos durante la implementación**  
A lo largo del trabajo el grupo se topó con problemas de distinta índole, siendo en su mayoría los generados por la gramática.  
Durante la generación del Parser, surgió un conflicto reduce/reduce dado que nos enfrentábamos a un problema con la lista de parámetros informada dentro de un llamado a función. Este problema se debe a que la lista de parámetros dentro de una función puede estar vacía, por lo que se debe añadir en la gramática la posibilidad de que esto ocurra. Se solucionó añadiendo a la declaración de la llamada a función, ID '('')', y de esta forma se aceptan tanto parámetros, lista de parámetros ó ningún parámetro.  
Otro fue la aparición de un shift/reduce, provocado en las sentencias IF y ELSE dado que llegado a ese punto la gramática no especificaba que camino escoger. Se solucionó dando precedencia a una sentencia sobre la otra.