Diseño de Compiladores 1
Trabajo Practico 1 y 2

Autores:
Roman Luna
romanluna1234@gmail.com

Savo Gabriel Koprivica
skoprivica@alumnos.exa.unicen.edu.ar

Grupo:
20

Cátedra:
Diseño de compiladores 1

Profesores:
Jose A. Fernandez Leon

Introducción

El presente informe exhibe el desarrollo de la creación de un compilador basado en el lenguaje Java. En este informe se verá la creación del analizador Léxico y el analizador Sintáctico. Las decisiones tomadas y cómo resolvimos diversos problemas o cuestiones que nos fueron surgiendo.

Temas Particulares

6. Enteros largos (32 bits): Constantes enteras con valores entre -2^{31} y $2^{31} - 1$. Estas constantes llevarán el sufijo "**_l**".

Enteros sin signo (16 bits): Constantes con valores entre 0 y $2^{16} - 1$. Estas constantes llevarán el sufijo "**_ui**". Se deben incorporar a la lista de palabras reservadas las palabras **LONG** y **UINT**.

7. Punto Flotante de 32 bits: Números reales con signo y parte exponencial. La parte exponencial puede estar ausente. Si está presente, el exponente comienza con la letra "**e**" (mayúscula o minúscula) y el signo del exponente es obligatorio.

Puede estar ausente la parte entera o la parte decimal, pero no ambas. El „." es obligatorio. Ejemplos válidos:

1. .6 -1.2 3.e–5 2.E+34 2.5E-1 15. 0. 1.2e+10

Considerar el rango 1.17549435E-38 < x < 3.40282347E+38 U

-3.40282347E+38 < x < -1.17549435E-38 U 0.0

10. En los lugares donde un identificador puede utilizarse como operando (expresiones aritméticas o comparaciones), considerar el uso del operador '--' luego del identificador. Por ejemplo: a = b-- * 7_i,

z = a---b--,

IF (a-- > 2_l) …

13 .WHILE (<condicion>) **DO** <Bloque_de_sentencias_ejecutables>

<condicion> tendrá la misma definición que la condición de las sentencias de selección.

<Bloque_de_sentencias_ejecutables> podrá contener una sentencia, o un grupo de sentencias ejecutables delimitadas por llaves.

18. Herencia por Composición - Uso con nombre

19. Se dara en el Trabajo Practico N°3

21. Forward declaration

23. Se dara en el Trabajo Practico N°3

26. Se dara en el Trabajo Practico N°3

29. Conversiones Explícitas: **TOF**

33. Comentarios multilínea: Comentarios que comiencen con “*{” y terminen con “}*” (estos comentarios pueden ocupar más de una línea).

35. Cadenas multilínea: Cadenas de caracteres que comiencen y terminen con “ % ” . Estas cadenas pueden ocupar más de una línea. (En la Tabla de símbolos se guardará la cadena sin los saltos de línea). Ejemplo: % ¡Hola
mundo! %

Analizador Léxico

Decidimos usar el lenguaje Java para realizar nuestro compilador.

Tenemos los siguientes paquetes:

- Acciones Semánticas: En este paquete se encuentran todas las acciones semánticas utilizadas para poder construir los token de manera correcta. Para implementarla decidimos utilizar una interfaz, la cual implementamos en cada AS individual.

  Las acciones semánticas son:
  - AS0: Se encarga de leer el siguiente carácter en el programa
  - AS1: Lee el siguiente carácter y lo concatena con el carácter actual
  - AS2: Se encarga de leer el siguiente carácter y devolver el token del literal
  - AS3: Se encarga de verificar que el identificador no supera el rango máximo, lo agrega a la tabla de símbolos en caso de no estar y retorna el token
  - AS4: Elimina el último carácter y retorna el token
  - AS5: Verifica de qué tipo de constante entera se trata, verifica que su rango sea el correcto, lo agrega a la tabla de símbolos en caso de no estar y retorna el token
  - AS6: Verifica si la constante flotante no excede el rango, si está en la tabla de símbolos y retorna el token
  - AS7: Lee el siguiente carácter, lo concatena con el carácter actual y retorna el token
  - AS8: Acción semántica encargada de leer las palabras reservadas
  - AS9: Encargada de leer los comentarios y descartarlos
  - AS10: Encargada de leer cadenas
  - ASE: Acción semántica encargada de retornar error
- archivosTxt: En este paquete se encuentran los archivos txt de los cuales generamos las diferentes tablas/matrices que vamos a necesitar para el funcionamiento de nuestro analizador léxico

- Compilador: Este paquete contiene todas las clases que se utilizan para el funcionamiento del Compilador. Las cuales detallaremos posteriormente ne el informe
- Testeos: Contiene todos los códigos de prueba utilizados, tanto correctos como con errores.

Clases

Tabla de símbolos

Esta clase es la encargada de almacenar los diferentes símbolos que van surgiendo en el código que estemos compilando. Los símbolos se guardan en un Hashtable<string, símbolo> en el cual la llave string corresponde al lexema. Antes de agregar un nuevo símbolo corrobora que no exista previamente en la tabla.

Símbolo

En esta etapa del trabajo un símbolo solo tiene un Lexema y un id asociado. En entregas posteriores se le agregara un tipo y un ámbito

LectorArchivo

Esta clase la decidimos implementar para mantener un mejor orden en el compilador. Dicha clase se encarga de crear las tablas o matrices correspondientes a partir de los archivos txt mencionados anteriormente Como por ejemplo la Matriz de transición estados, matriz de acciones semánticas, caracteres ascii, etc.

Constantes

Como Lector Archivo, esta clase la decidimos crear para mejorar la legibilidad y el orden del código. En esta se encuentran constantes que utilizamos en varias partes del programa y a su vez las tablas/matrices que vamos a utilizar, como la Matriz de acciones semánticas o las palabras reservadas, etc.

Main

El main es el encargado de realizar la ejecución del parser, el cual utiliza el léxico. También es la encargada de luego mostrar los errores sintácticos y tokens detectados por el analizador léxico.

AnalizadorLexico

Esta es la clase principal del analizador léxico, la cual se encarga de generar los tokens y enviárselos al parser.

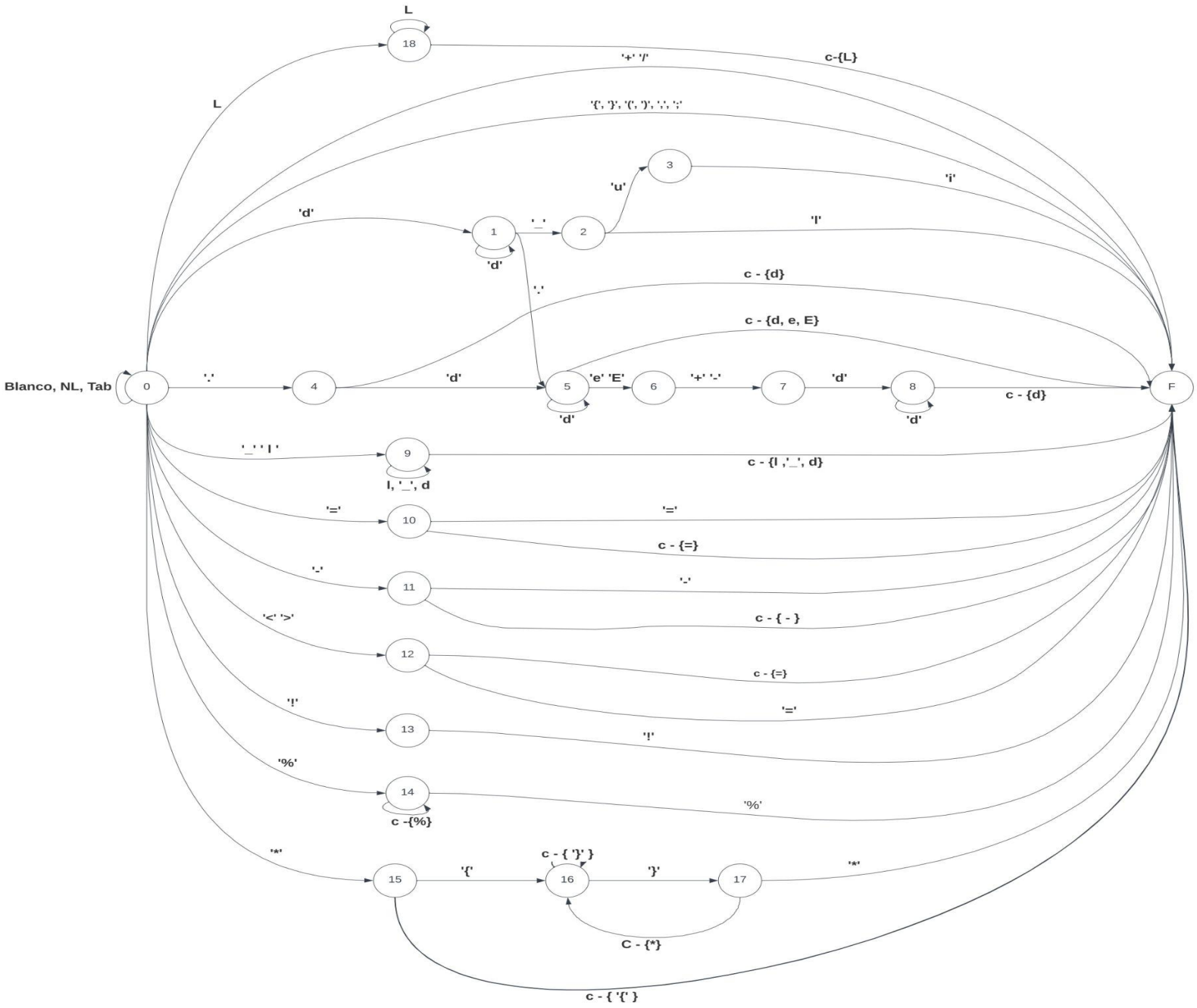
Matriz de Acciones Semánticas:

	BL	TAB	NL	I	'_'	D	'e'	'E'	'+'	'-'	'*'	'/'	'='	'!'	'<'	'>'	'.'	';'	','	PR	'{'	'}'	'%'	'('	')'	'l'	'u'	'i'	OTRO
0	AS0	AS0	AS0	AS1	AS1	AS1	AS1	AS1	AS2	AS1	AS1	AS2	AS1	AS1	AS1	AS1	AS1	AS2	AS2	AS1	AS2	AS2	AS1	AS2	AS2	AS1	AS1	AS1	ASE
1	ASE	ASE	ASE	ASE	AS1	AS1	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	AS1	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE
2	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	AS5	AS1	ASE	ASE
3	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	AS5	ASE
4	AS4	AS4	AS4	AS4	AS4	AS1	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	ASE	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	ASE
5	AS6	AS6	AS6	AS6	AS6	AS1	AS1	AS1	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	ASE
6	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	AS1	AS1	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE
7	ASE	ASE	ASE	ASE	ASE	AS1	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE
8	AS6	AS6	AS6	ASE	ASE	AS1	ASE	ASE	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	AS6	ASE
9	AS3	AS3	AS3	AS1	AS1	AS1	AS1	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS3	AS1	AS1	AS1	ASE
10	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	ASE	ASE	AS4	AS4	AS7	ASE	ASE	ASE	AS4	ASE	ASE	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	ASE
11	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS7	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	ASE
12	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS7	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	ASE
13	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	AS7	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE	ASE
14	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS10	AS1	AS1	AS1	AS1	AS1	ASE
15	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS4	AS1	AS4	AS4	AS4	AS4	AS4	AS4	AS4	ASE
16	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	ASE
17	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS9	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	ASE
18	AS8	AS8	AS8	AS8	AS1	AS8	AS8	AS1	AS8	AS8	AS8	AS8	AS8	AS8	AS8	AS8	AS8	AS8	AS8	AS1	AS8	AS8	AS8	AS8	AS8	AS8	AS8	AS8	ASE

Matriz de transición de estados:

	BL	TAB	NL	I	'_'	D	'e'	'E'	'+'	'-'	'*'	'/'	'='	'!'	'<'	'>'	'.'	';'	','	PR	'{'	'}'	'%'	'('	')'	'l'	'u'	'i'	OTRO
0	0	0	0	9	9	1	9	18	0	11	15	0	10	13	12	12	4	0	0	18	0	0	14	0	0	9	9	9	-1
1	-1	-1	-1	-1	2	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	3	-1	-1
3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1
4	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	-1
5	0	0	0	0	0	5	6	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
6	-1	-1	-1	-1	-1	-1	-1	-1	7	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
7	-1	-1	-1	-1	-1	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	0	0	0	-1	0	8	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
9	0	0	0	9	9	9	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	9	9	-1
10	0	0	0	0	0	0	0	0	-1	-1	0	0	0	-1	-1	-1	0	-1	-1	0	0	0	0	0	0	0	0	0	-1
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
13	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	0	14	14	14	14	14	-1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	-1
16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	17	16	16	16	16	16	16	-1
17	16	16	16	16	16	16	16	16	16	16	0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	-1
18	0	0	0	0	18	0	0	18	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	-1

Autómata De transición de Estados(link:https://lucid.app/lucidchart/5c4ef08f-0410-41d1-bbaf-ababff8bef52/edit?invitationId=inv_f167d908-d5f9-4819-941f-a7754df7bc3b&page=0_0#)



Consideración de Errores Léxicos

Consideramos como errores a cualquier carácter que no pertenezca a nuestro lenguaje y también a las constantes fueras de rango. Consideramos Warnings a los identificadores fuera de rango(avisa el warning y los recorta para que sean aceptados).

Analizador Sintactico

El objetivo de esta parte del trabajo era construir un Parser (Analizador Sintáctico) que invoque al Analizador Léxico creado en el Trabajo Práctico Nº 1, y que reconozca un lenguaje con las siguientes características:

Programa constituido por un conjunto de sentencias, que pueden ser declarativas o ejecutables. Las sentencias declarativas pueden aparecer en cualquier lugar del código fuente, exceptuando los bloques de las sentencias de control.

Los elementos declarados sólo serán visibles a partir de su declaración (esto será chequeado en etapas posteriores).

El programa estará delimitado por llaves ‘{‘ y ‘}’. Cada sentencia debe terminar con coma ",".

Decisiones e implementación:

Para realizar el parser tuvimos que implementar toda la gramática siguiendo las pautas que nos brinda el TP2. Tratamos de llevar un orden en la gramática para facilitar su entendimiento y a su vez utilizar nombres claros a cada regla. Decidimos agregar reglas con errores los cuales son almacenados en una lista “errores” utilizando el método “agregarErrores” para ser mostrados una vez finalizada la compilación del código.

Los códigos de prueba utilizados tienen el siguiente formato:

```
_p {

    FLOAT _numero1,

    IF(10_l > 8_l){
        _numero2 = _numero1,
        _hola = _chau,
    }ELSE{
        _perro = 10_l,
    } END_IF,

    _hola = 10_l,

    IF(_hola == 10_l){
            _f1 ( _a),
        IF(10_l >= 9_l){
            _hola = _esto,
        }ELSE{
            _hola = _chau,
        }END_IF,

    }END_IF,

}
```

Al ser ejecutados por el parser, el resultado es:

```
Linea: 3. Se reconocio un tipo FLOAT
Linea: 3. Se reconocio una declaracion de variables
Linea: 3. Se reconocio sentencia declarativa
Linea: 5. Se reconocio una constante
Linea: 5. Se reconocio una constante
Linea: 5. Se reconocio una comparacion por mayor
Linea: 6. Se reconocio un Identificador
Linea: 6. Se reconocio una asignacion
Linea: 6. Se reconocio sentencia Ejecutables
Linea: 7. Se reconocio un Identificador
Linea: 7. Se reconocio una asignacion
Linea: 7. Se reconocio sentencia Ejecutables
Linea: 9. Se reconocio una constante
Linea: 9. Se reconocio una asignacion
Linea: 9. Se reconocio sentencia Ejecutables
Linea: 10. Se reconocio un IF_ELSE
Linea: 10. Se reconocio sentencia ejecutable
Linea: 12. Se reconocio una constante
Linea: 12. Se reconocio una asignacion
Linea: 12. Se reconocio sentencia Ejecutables
Linea: 12. Se reconocio sentencia ejecutable
Linea: 14. Se reconocio un Identificador
Linea: 14. Se reconocio una constante
Linea: 14. Se reconocio una comparacion de igualdad
Linea: 15. Se reconocio un Identificador
Linea: 15. Se reconocio una invocacion a funcion con parametro
Linea: 16. Se reconocio una constante
Linea: 16. Se reconocio una constante
Linea: 16. Se reconocio una comparacion por mayor o igual
Linea: 17. Se reconocio un Identificador
Linea: 17. Se reconocio una asignacion
Linea: 17. Se reconocio sentencia Ejecutables
Linea: 19. Se reconocio un Identificador
Linea: 19. Se reconocio una asignacion
Linea: 19. Se reconocio sentencia Ejecutables
Linea: 20. Se reconocio un IF_ELSE
Linea: 22. Se reconocio un IF
```

mostrando la Línea en la que se reconoció una regla y que fue lo reconocido.

Una vez finalizada la ejecución del parser, los errores que se encontraron son mostrados. Se imprime la línea y una descripción del error sintáctico encontrado:

```
Linea 84, Se reconocio el programa
ARRANCO EL PROGRAMA
Linea: 16. Error sintactico . Falta RETURN en la funcion
Linea: 21. Error sintactico . Falta '}' en la funcion
Linea: 26. Error sintactico . Falta '{' en la funcion
Linea: 31. Error sintactico . Falta ')' en la funcion
Linea: 36. Error sintactico . Falta '(' en la funcion
Linea: 41. Error sintactico . Falta nombre en la funcion
Linea: 46. Error sintactico . Falta la palabra VOID en la funcion
Linea: 53. Error sintactico . Falta RETURN en la funcion
Linea: 58. Error sintactico . Falta '}' en la funcion
Linea: 63. Error sintactico . Falta '{' en la funcion
Linea: 68. Error sintactico . Falta ')' en la funcion
Linea: 73. Error sintactico . Falta '(' en la funcion
Linea: 78. Error sintactico . Falta nombre en la funcion
Linea: 83. Error sintactico . Falta la palabra VOID en la funcion
```

Lista de no terminales de la gramática:

- nombre programa
- bloque_sentencia_programa
- sentencia_programa
- sentencias_declarativas
- sentecias_ejecutables
- declaracion_variables
- declaracion_funciones
- tipo
- list_de_varaibles
- cuerpo_de_funcion
- sentencia_declarativa_especifica
- sentencias_retorno
- declaracion_clases
- cuerpo_clase
- list_objts_clase
- sentencia_ejecucion_funcion
- sentencia_asignacion
- valor_asignacion
- invocacion_funcion
- sentencia_IF
- condicion_if_while
- bloque_sentencias_ejecutables
- sentencias_salida
- sentencias_control
- sentencias_while_do
- expr_aritmetic
- termino
- factor
- const

Errores y consideraciones en estas etapas

- En estas etapas tuvimos problemas al querer identificar el rango de las constantes. Al ser el rango distinto para números positivos y negativos no sabíamos si esto era necesario verificarlo en el analizador sintáctico o léxico. En el léxico las variables LONG no era posible comprobar el rango ya que las variables de tipo int solo admiten un rango de (2^31)-1 por lo que el último número negativo no lo íbamos a poder verificar en el analizador sintáctico por lo que utilizamos un valor de prueba en esta primera etapa.
- Dentro de las funciones solo es posible realizar sentencias declarativas ya que el enunciado no menciona nada de sentencias ejecutables.
- En el menu es posible que al elegir varios archivos txt en una ejecucion se rompa. Solo hay que reiniciar y volver a ejecutar el jar.
- Es necesario que los archivos txt sean UNIX(LF).

Conclusiones

Durante el desarrollo de estas dos etapas pudimos comprender el funcionamiento del léxico y el sintáctico de un compilador. Como un compilador analiza un código y a partir de este genera tokens e identifica sentencias que luego van a ser usadas en etapas posteriores. Como son detectados los errores léxicos y sintácticos y una idea general de cómo funciona un compilador hasta este punto.