Trabajo Práctico
Diseño de Compiladores

Analizador Léxico y Sintáctico

Grupo n° 18
Integrantes:

Petrucelli, Augusto.
oapetruccelli@gmail.com
Rojas, Emiliano.
erojas@alumnos.exa.unicen.edu.ar
Velez, Ezequiel.
eze.5.94@gmail.com

Docente Asignado:

Dazeo, Nicolás.

Índice:

Introducción

El presente informe servirá de reseña que abarcará todos los detalles del desarrollo de los trabajos prácticos 1 y 2 para Diseño de Compiladores.
Esta primera parte del compilador comprende el diseño e implementación de un Analizador Léxico y una gramática que sean capaces de leer un código fuente como entrada en un lenguaje asignado por la cátedra, informando los errores encontrados en el mismo.

Analizador Léxico

El analizador léxico se encarga de verificar que se cumplan las reglas léxicas del código fuente, y agrupar los caracteres en unidades significativas llamadas tokens.
Los tokens que debe reconocer son los siguientes:

- Identificadores cuyos nombres pueden tener hasta 25 caracteres de longitud. El primer carácter debe ser un “_”,y el resto pueden ser letras y dígitos. Los identificadores con longitud mayor deberán ser informados como error, y el token erróneo deberá ser descartado. Las letras utilizadas en los nombres de identificador pueden ser mayúsculas o minúsculas, y el lenguaje será case sensitive. Entonces, el identificador MyVariable, no será igual a myvariable.
- Constantes correspondientes a los temas particulares asignados a cada grupo. Las constantes fuera de rango deberán ser tratadas con la técnica de reemplazo, utilizando para ello, el valor más grande dentro del rango permitido. Tal situación se informará como Warning.
- Operadores aritméticos: “+”, “-” ,“*”, “/”.
- Operador de asignación: “:=”
- Comparadores: “>=”, “<=”, “>”, “<”, “=”, “!=”
- “(” “)” “{” “}” “,” y “;”
- Cadenas de caracteres correspondientes al tema particular de cada grupo.
- Palabras reservadas (en minúsculas): If, else, end_if, print y demás símbolos / tokens indicados en los temas particulares asignados al grupo.

El Analizador Léxico debe eliminar de la entrada (reconocer, pero no informar como tokens al Analizador Sintáctico), los siguientes elementos.

- Comentarios correspondientes al tema particular de cada grupo.
- Caracteres en blanco, tabulaciones y saltos de línea, que pueden aparecer en cualquier lugar de una sentencia..

Además, como medida de control, guarda el numero de linea en el que se trabaja.

Temas Particulares del Grupo:

1. Enteros: Constantes enteras con valores entre -2^{15} y $2^{15} - 1$. Estas constantes llevarán el sufijo "_i". Se debe incorporar a la lista de palabras reservadas la palabra integer.
3. Enteros largos: Constantes enteras con valores entre -2^{31} y $2^{31} - 1$. Estas constantes llevarán el sufijo "_l". Se debe incorporar a la lista de palabras reservadas la palabra linteger.
10. Incorporar a la lista de palabras reservadas, las palabras case y do.
12. Incorporar a la lista de palabras reservadas, las palabras void, fun y return.
18. Comentarios multilínea: Comentarios que comiencen con "#" y terminen con "#" (estos comentarios pueden ocupar más de una línea).
19. Cadenas de 1 línea: Cadenas de caracteres que comiencen y terminen con " ' " (estas cadenas no pueden ocupar más de una línea).

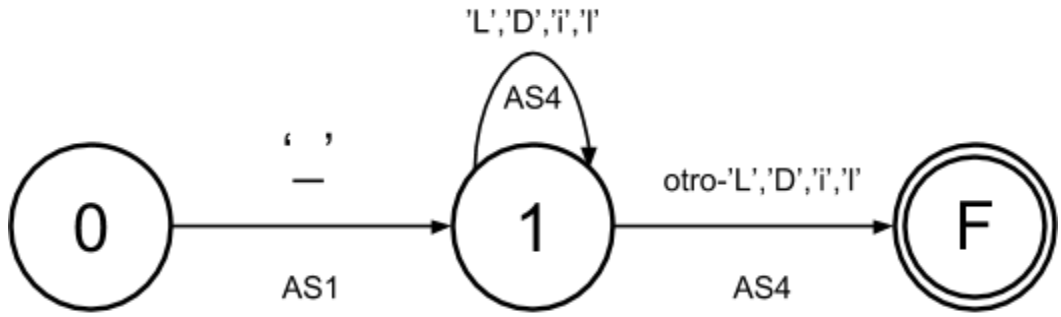
Decisiones de diseño e implementación

El analizador léxico fue implementado siguiendo el paradigma de la programación orientada a objetos.
La clase Lex recibe la ruta, a un archivo de texto contenedor del código fuente, como primer argumento de la aplicación en la línea de comandos. Esta clase será utilizada para obtener uno a uno los tokens reconocidos en el texto. Su método yylex() es el que retorna un token al ser invocado.
Se escribió una gramática, utilizando SublimeText, siguiendo la sintaxis especificada para YACC, según las indicaciones para la instancia asignada del trabajo práctico.
El compilador fue programado en Java, utilizando el IDE eclipse.

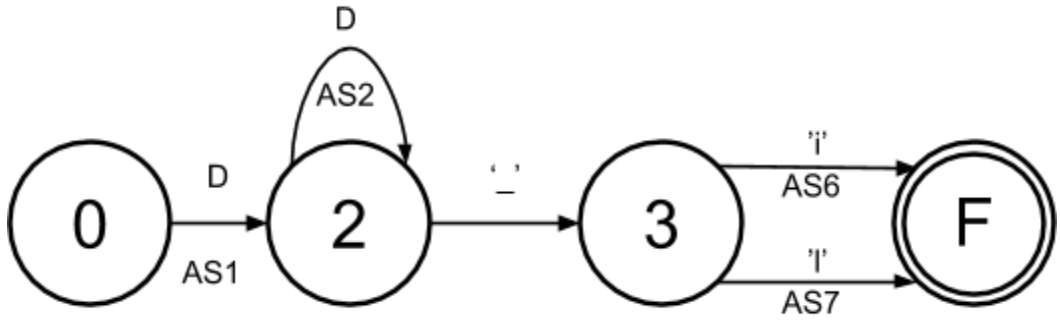
Diagrama de transición de estados

A continuación, se encuentran las consideraciones tenidas en cuenta para realizar el diagrama de transición de estados, junto con la porción de diagrama que se corresponde:
➔ 'L' representa las letras excepto la 'i' y 'l' que tienen un tratamiento especial por formar parte de las constantes enteras y enteras largas.
➔ 'D' representan los dígitos del 0 al 9.
➔ 'otro - ' significa cualquier carácter exceptuando los que se encuentran detrás del '-'.

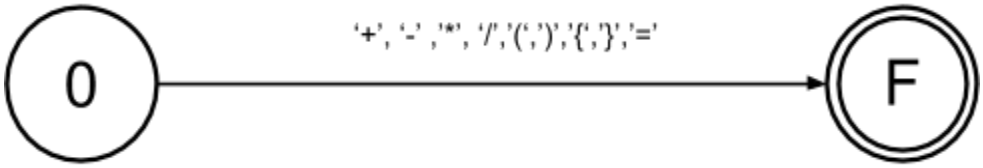
- Identificadores



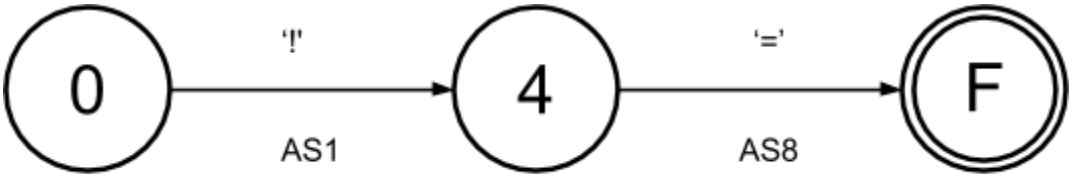
- Constantes



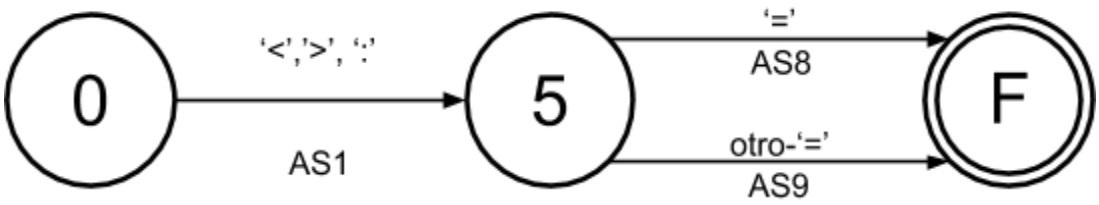
- Operadores aritméticos, paréntesis y llaves, comparador de igualdad



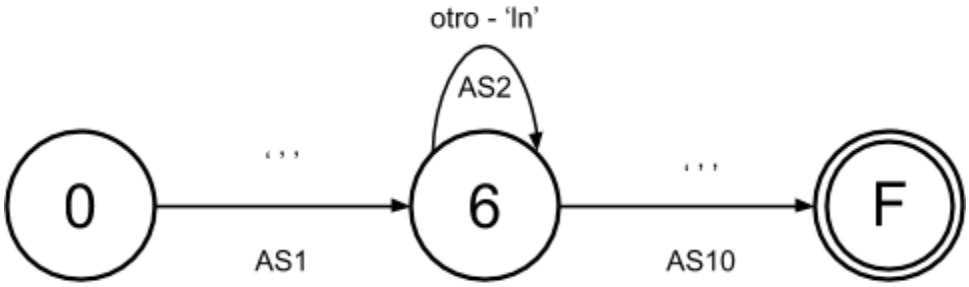
- comparador “Not Equal”



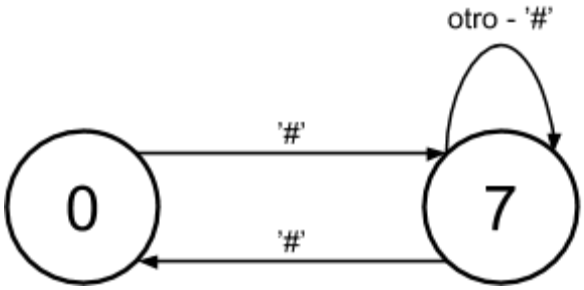
- Comparadores Menor, Menor o Igual, Mayor, Mayor o Igual



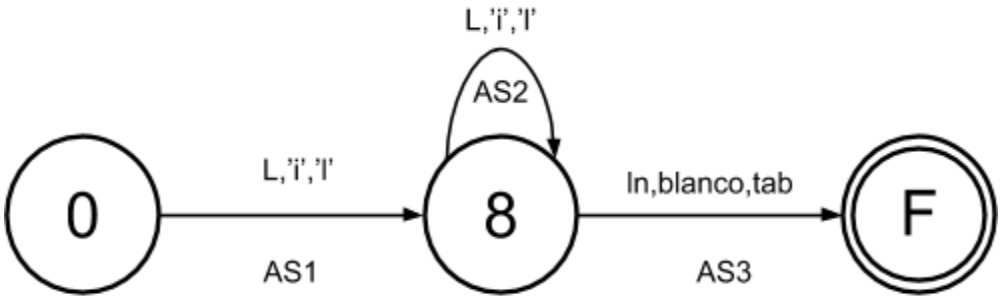
- Cadena Monolínea



- Comentario Multilínea



- Palabras Reservadas



Matriz de Transición de estados

	L	D	+ - * / ( ) { } , ;	!	=	< > :	_	i	I	'	#	blanco/ tab	/n /r	EOF
0	8	2	F	4	F	5	1	8	8	6	7	0	0	F
1	1	1	F	F	F	F	F	1	1	F	F	F	F	F
2		2					3							
3								F	F					
4					F									
5	F	F	F	F	F	F	F	F	F	F	F	F	F	F
6	6	6	6	6	6	6	6	6	6	F	6	6		
7	7	7	7	7	7	7	7	7	7	7	0	7	7	
8	8	F	F	F	F	F	8	8	8	F	F	F	F	F

Tabla de Acciones Semánticas

	L	D	+ - * / ( ) { } , ;	!	=	< > :	_	i	I	'	#	blanco/ tab	/n /r	EOF
0	AS1	AS1	AS8	AS1	AS8	AS1	AS1	AS1	AS1	AS1				-
1	AS4	AS4	AS5	AS5	AS5	AS5	AS5	AS4	AS4	AS5	AS5	AS5	AS5	AS5
2		AS2					-							
3								AS6	AS7					
4					AS8									
5	AS9	AS9	AS9	AS9	AS8	AS9	AS9	AS9	AS9	AS9	AS9	AS9	AS9	AS9
6	AS2	AS2	AS2	AS2	AS2	AS2	AS2	AS2	AS2	AS10	AS2	AS2		
7	-	-	-	-	-	-	-	-	-	-	-	-	-	
8	AS2	AS3	AS3	AS3	AS3	AS3	AS2	AS2	AS2	AS3	AS3	AS3	AS3	AS3

Color Coding:

Error: no permitido	Palabras reservadas
Estado Completado	Asignacion/Not Equal
Saltar caracteres	Menor, Menor Igual, Mayor, Mayor Igual
Identificadores	Cadena de caracteres
Entero/Entero Largo	Comentario

Acciones Semánticas

AS1

1. Inicializar string
2. Agregar letra al string

AS2

1. Agregar letra o dígito al string

AS3

1. Devolver a la entrada el último carácter leído
2. Buscar en la TS
3. Si está,
    a. Devolver la Palabra Reservada
4. Si no está,
    a. Informar error

AS4

1. Verificar longitud del string (25)
2. Si cumple el límite
    a. Agregar letra o dígito al string

AS5

1. Devolver a la entrada el último carácter leído
2. Buscar en la TS
3. Si no está,
    a. Alta en la TS
4. Devolver ID + Punt TS

AS6

1. Transformar a entero
2. Comprobar límite
3. Alta en la TS
4. Devolver CTE + Punt TS

AS7

1. Transformar a entero largo
2. Comprobar límite
3. Alta en la TS
4. Devolver CTE + Punt TS

AS8

1. Inicializar string
2. Agregar letra al string
3. Buscar en la TS
4. Si está,
    a. Devolver la Palabra Reservada
5. Si no está,
    a. Informar error

AS9

1. Devolver el último caracter.

AS10

1. Agregar caracter
2. Buscar en la TS
3. Si no está,
    a. Alta en la TS
4. Devolver STR + Punt TS

Implementación de las matrices

Para su implementación, se utilizaron dos matrices de enteros, los cuales representan los cambios de estado para cada carácter leído. Para las diferentes acciones semánticas se creó la clase abstracta AccionSemantica, con el método abstracto ejecutar, el cual realiza una acción particular en cada caso.

Tabla de Símbolos

En la tabla de símbolos se almacenan identificadores, constantes y cadenas de caracteres, representados por el lexema. Se implementó con una estructura dinámica de hash que, dado un string usado como clave, devuelve una lista de los atributos de esa variable.
Las palabras reservadas del lenguaje son precargadas a la tabla.
Actualmente se guarda el tipo de token, para distinguirlos entre ellos, y el número asociado.

Analizador Sintáctico

El analizador sintáctico verifica el cumplimiento de las reglas sintácticas. Solicita tokens al analizador léxico y verifica que estos respeten la gramática del lenguaje, construyendo el árbol de análisis, además de informar errores sintácticos. Se implementó una gramática mediante la sintaxis especificada para YACC, una herramienta que se encarga de realizar el árbol de parsing de forma automática. Se construyó un parser que invoca al analizador léxico creado anteriormente, que reconoce un lenguaje que incluye:

- Programa: constituido por sentencias declarativas o ejecutables.
- Sentencias declarativas: declaración de datos para los tipos de datos INTEGER y LINTEGER, con la siguiente sintaxis: <lista_de_variables>:<tipo>, donde las variables de la lista se separan con punto y coma.

Sentencias ejecutables:

- -Cláusula de selección (IF)
- -Sentencia de salida de mensajes por pantalla (PRINT)
- -Bloques de sentencias delimitadas por llaves
- -Sentencia de control CASE-DO
- -Asignaciones

Conversiones explícitas (INTEGER->LINTEGER)
Declaración y llamado de funciones (FUN, VOID)

Se definió una gramática mediante la sintaxis de YACC, para servir de estructura a seguir al analizador sintáctico. La gramática implementada debía seguir la siguiente sintaxis:
%{
}%
DECLARACIONES
%%
REGLAS
%%
CÓDIGO
La sección de declaraciones es donde pueden definirse los tokens, precedencias, entre otros, que se van a utilizar en las reglas. En la sección de reglas se genera la gramática. Durante su construcción, se respetaron las recursividades a izquierda, para reducir los problemas shift/reduce y reduce/reduce que pueden surgir. Por la forma en que fue definida la gramática no fue necesaria la asignación de precedencia a las reglas.

Errores Sintácticos

Los errores sintácticos que contempla la gramática son los casos más propensos a tener errores en la declaración de las sentencias. Para cada caso, se informa la línea del error junto con una breve descripción.

Consideraciones

-Como el código puede contener valores de enteros y enteros largos negativos se optó por usar un contador de referencias en las constantes positivas, ya que el léxico solo toma en cuenta valores positivos. Recién en la gramática se distingue el signo, pero no hay que perder el hecho de que un mismo número puede ser tanto positivo como negativo.
-Como salida del programa, se muestra por pantalla todas las diferentes impresiones de cada analizador, marcando entre paréntesis - (AL),(AS),(Parser) - A qué sección del programa corresponde.

Conclusiones

En cuanto al desarrollo de la primera parte del compilador, podemos decir que la etapa del analizador léxico se encuentra ya finalizada. En cuanto a la relación entre el léxico y el parser, el Analizador Sintáctico va pidiendo tokens al Léxico, es decir es un compilador Monolítico, ya que la ejecución del programa utiliza principalmente la clase Parser, definida por el YACC/Java en el desarrollo del analizador sintáctico.
Una vez que el compilador este completo, las dos etapas involucradas en este trabajo se comunicarán con la generación intermedia de código y, posteriormente, assembler, de cualquier programa escrito en nuestro lenguaje. Por lo tanto una buena definición del

analizador sintáctico es esencial para recibir el código assembler correcto del programa a compilar.