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.

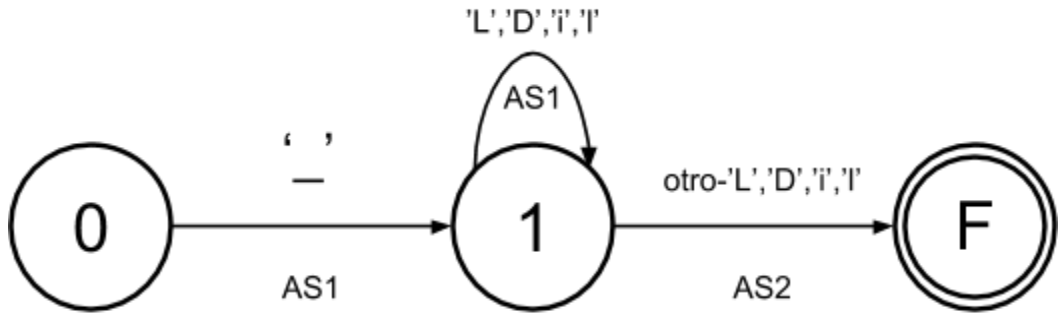
Temas Particulares del Grupo:

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).

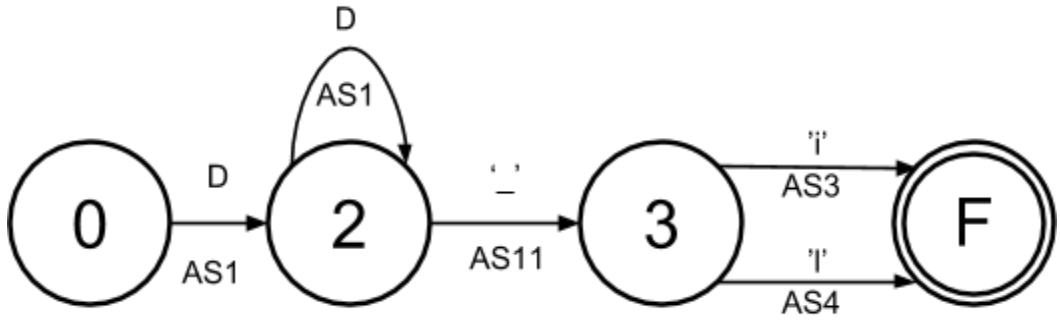
Diagrama de transición de estados

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 '-'.

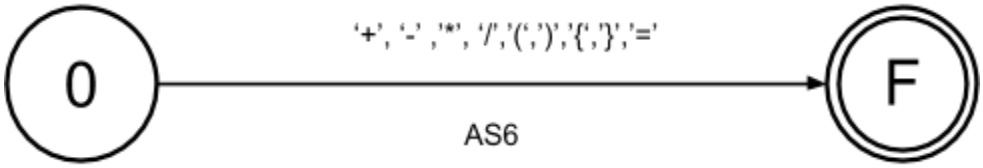
- Identificadores



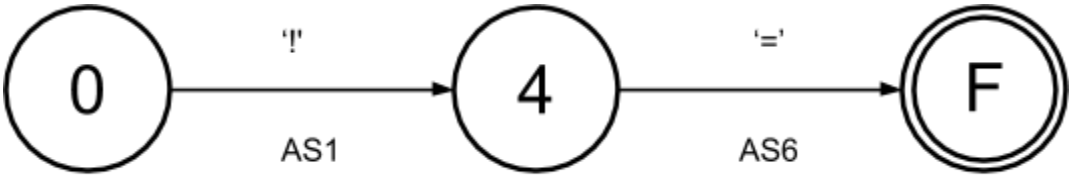
- Constantes



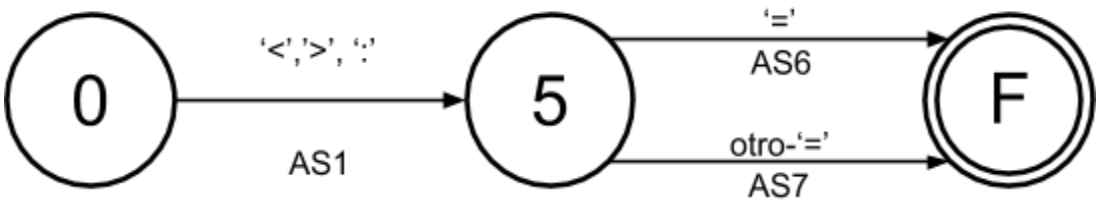
- Operadores aritméticos, paréntesis y llaves, comparador de igualdad



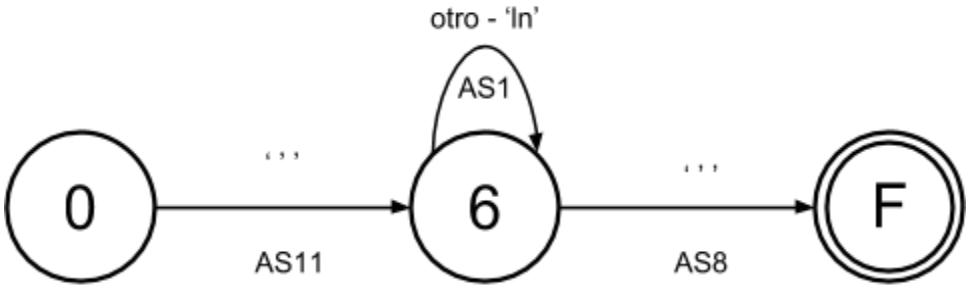
- comparador “Not Equal”



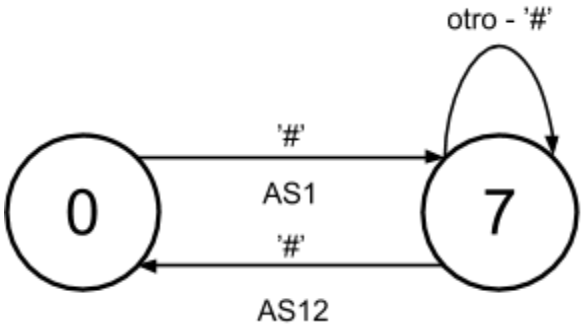
- Comparadores Menor, Menor o Igual, Mayor, Mayor o Igual



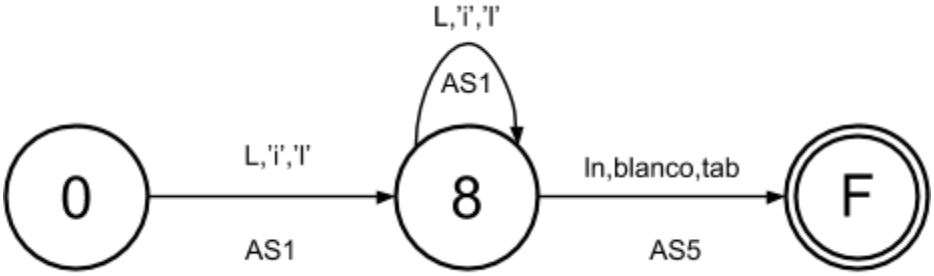
- Cadena Monolínea



- Comentario Multilínea



- Palabras Reservadas



Matriz de Transición de estados

	L	D	+ - * / ( ) { } , ;	!	=	: < >	_	i	I	'	#	\s \t	\n	otro	EOF
0	8	2	F	4	F	5	1	8	8	6	7	0	0	F	F
1	1	1	F	F	F	F	F	1	1	F	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	F
6	6	6	6	6	6	6	6	6	6	F	6	6		6	
7	7	7	7	7	7	7	7	7	7	7	0	7	7	7	
8	8	F	F	F	F	F	8	8	8	F	F	F	F	F	F

Tabla de Acciones Semánticas

	L	D	+ - * / ( ) { } , ;	!	=	: < >	_	i	I	'	#	\s \t	\n	otro	EOF
0	AS1	AS1	AS6	AS1	AS6	AS1	AS1	AS1	AS1	AS11	AS1	AS11	AS10	AS11	AS11
1	AS1	AS1	AS2	AS2	AS2	AS2	AS2	AS1	AS1	AS2	AS2	AS2	AS2	AS2	AS2
2	AS9	AS1	AS9	AS9	AS9	AS9	AS11	AS9	AS9	AS9	AS9	AS9	AS9	AS9	AS9
3	AS9	AS9	AS9	AS9	AS9	AS9	AS9	AS3	AS4	AS9	AS9	AS9	AS9	AS9	AS9
4	AS9	AS9	AS9	AS9	AS6	AS9	AS9	AS9	AS9	AS9	AS9	AS9	AS9	AS9	AS9
5	AS7	AS7	AS7	AS7	AS6	AS7	AS7	AS7	AS7	AS7	AS7	AS7	AS7	AS7	AS7
6	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS8	AS1	AS1	AS9	AS1	AS9
7	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS1	AS12	AS1	AS1	AS1	AS9
8	AS1	AS5	AS5	AS5	AS5	AS5	AS1	AS1	AS1	AS5	AS5	AS5	AS5	AS5	AS5

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. Agregar caracter al string

AS2

1. Verificar longitud del string (25)
2. Si no cumple el límite truncar
3. Agregar el identificador a la TS
4. Retornar Token

AS3

1. Transformar a entero
2. Comprobar límite
3. Alta en la TS
4. Retornar Token

AS4

1. Transformar a entero largo
2. Comprobar límite
3. Alta en la TS
4. Retornar Token

AS5

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

AS6

1. Agregar el último caracter
2. Retornar Operador

AS7

1. Devolver el último caracter
2. Retornar Operador

AS8

1. Agregar la cadena a la TS
2. Retornar Token

AS9

1. Informar Caracter Invalido.

AS10

1. Incrementar el contador de línea

AS11

Accion Nula

AS12

1. Descartar Comentario

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.

Diseño de implementación

Para el desarrollo y la implementación del analizador léxico y sintáctico se utilizó el lenguaje de programación Java en el entorno de desarrollo Eclipse. Para clarificar el código se dividió el problema en diferentes partes: por un lado, la implementación de las estructuras utilizadas tanto por el analizador léxico como el sintáctico y que son globales al proceso de compilación (Tabla de símbolos); por otro lado, las clases y la funcionalidad propia del analizador léxico; la lista de acciones semánticas que son invocadas por éste; y, finalmente las tareas y responsabilidades del analizador sintáctico, que serán detalladas más adelante.

A continuación se describen brevemente cada uno de estos apartados indicando las decisiones de diseño tenidas en cuenta en cada caso particular.

Tabla de Simbolos

La tabla de símbolos almacena un registro para cada identificador utilizado del código fuente. Durante la primera fase de la compilación, cada vez que el analizador léxico detecta un token de tipo identificador, constante o cadena de caracteres lo ingresa en esta tabla, que luego será utilizada por el resto de las etapas.
Para la implementación de dicha estructura se utilizó una HashTable de tipo <String, Token>, donde la clave de tipo String representa el lexema del token, que hace referencia al valor de un identificador, una constante o cadena de caracteres. El valor de tipo “Token” contiene todos los atributos y características referentes a dicho elemento.
La elección de dicha estructura se justifica en la necesidad de contar con una organización dinámica que permita agregar nuevos símbolos a la tabla, así como también, actualizarlos o modificarlos de forma eficiente, ya que la gestión de dicha estructura consume gran parte del proceso de compilación.

Esta clase contiene también, los identificadores asociados a cada token del programa fuente. Además utilizamos dicha clase para guardar la lista de palabras reservadas aceptadas por el lenguaje con su respectivo valor entero identificador. Dado que los caracteres ASCII están numerados de 0 a 255, debimos comenzar a definir los identificadores de nuestros tokens a partir del valor 257, para evitar conflictos posteriormente con la herramienta que genera el parser.

Analizador Lexico

Para el diseño del analizador léxico se implementaron clases vinculadas con la funcionalidad del mismo. En primer lugar, se cuenta con una clase “*BufferLectura*” que simula el archivo que contiene el programa fuente a ser leído. Además, lleva el control del número de línea por la que

se va leyendo el archivo, que es modificado desde una acción semántica. De esta manera, dicho buffer es el encargado de leer y consumir los caracteres de entrada (caracteres ASCII), y entregarlos al léxico cada vez que éste lo solicite y hasta que se llegue al final del programa fuente.

Otra de las estructuras implementadas fue la de "*Token*", que contiene toda la información relevante para el token: su identificador, lexema, descripción y demás atributos que se quieran agregar posteriormente. Esta es la estructura que se almacena en la tabla de símbolos y la referencia a dicho elemento se guarda en la variable *yylval* del parser.

Finalmente, para la implementación del analizador léxico, se generó la clase "*AnalizadorLexico*", en la cual se utilizan las estructuras mencionadas anteriormente. Adicionalmente, se cuenta con una matriz de transición de estados, que se obtiene a partir del diagrama de transición de estados, el cual representa la estructura de los tokens del programa y definen las posibles transiciones de un estado a otro. A su vez, se posee una matriz de acciones semánticas que se asocian a cada transición y especifican una acción a ejecutar. Esta clase, adicionalmente, define la función "*yylex*", la cual se encarga de proveer tokens al analizador sintáctico cada vez que éste lo solicite. Por una cuestión de diseño, este método retorna un número entero que representa el identificador del token detectado en el programa y a su vez, asigna a la variable "*yylval*" del parser generado, el token devuelto por la acción semántica. Podríamos resumir la funcionalidad de *yylex* en los siguientes pasos:

1. Le solicita al "*BufferLectura*" el próximo caracter leído desde el programa fuente (la entrada). El buffer le retorna el código ASCII correspondiente al caracter leído.
2. Convierte el valor devuelto por el buffer al número de columna de la matriz que corresponde con esa entrada.
3. Aplica la acción semántica que se encuentra en la matriz de acciones semánticas en la celda [estado actual] [entrada]
4. Cambia de estado, según el valor que se encuentra en la celda [estado actual] [entrada] de la matriz de transición de estados.
5. Si se llega a estado final, se devuelve el identificador del token obtenido por la aplicación de la acción semántica, caso contrario continúa el proceso en el paso 1.

Acciones Semánticas

En cuanto a las acciones semánticas se creó una interfaz “*AccionSemantica*”, la cual define el método “*ejecutar*” que será implementado por cada acción semántica en particular, según lo que se debe hacer en cada transición. Es decir, cada celda de la matriz de acciones semánticas es una acción semántica que implementa dicha interfaz. De esta manera, cuando se desea aplicar la acción semántica correspondiente a una transición, dado el estado actual (que representa la fila de la matriz) y una entrada (la columna), se llama al método ejecutar de la acción semántica de dicha celda. La aplicación de la acción semántica, puede implicar diferentes efectos, desde aumentar el número de línea del programa fuente hasta verificar la validez de un identificador e ingresarlo en la tabla de símbolos. En todos los casos, la llamada al método ejecutar retorna un objeto de tipo “Token”, el cual puede ser nulo para determinadas acciones semánticas o ser un token listo a ser entregado al parser para otras, dependiendo siempre del estado en el que se encuentre el autómata y el caracter de entrada leído.

Analizador Sintactico

Lista de no terminales

- programa: es la regla inicial de la gramática, define que un programa se compone por un bloque de sentencias.
- bloque_sentencias: define las reglas sintácticas para poder definir una o más sentencias declarativas y/o ejecutables.
- bloque_control: define las reglas sintácticas para poder definir una o más sentencias ejecutables, es utilizado en las sentencias de control donde no puede haber sentencias declarativas. Dicho bloque puede ser una sentencia ejecutable (que puede o no estar entre llaves) o un más de una sentencia ejecutable entre llaves.
- sentencia_declarativa: define los tipos de sentencias declarativas que admite la gramática.
- funcion: define el tipo y dependiendo el tipo como debe ser la función.
- funcion _fun: define las reglas gramaticales de una función con retorno.
- funcion_void: define las reglas gramaticales de una función sin retorno.
- llamado_funcion: define la regla para poder llamar a las funciones.
- declaracion_variables: define las sentencias de declaración de variables.

- tipo: define los tipos que nos fueron asignados.
- sentencias_ejecutables: define los tipos de sentencias ejecutables que admite la gramática.
- sentencia_seleccion: define las reglas correspondiente a la sentencia de seleccion con y sin el ELSE.
- sentencia_control: define las reglas correspondientes a la sentencia se control asignada al grupo, CASE DO.
- sentencia_impresión: define la regla para imprimir una cadena de caracteres por pantalla utilizando la palabra reservada PRINT seguida de la cadena colocada entre paréntesis.
- condicion: define que la condición de la sentencia de selección está dada por expresión comparador expresión. Que la condición se encuentre entre paréntesis forma parte de la regla condicion_if.
- comparador: esta regla define los símbolos terminales que reconoce el lenguaje para realizar comparaciones. (<, <=, >, >=, !=, =). Los que tienen más de un caracter se encuentran expresados como Tokens.
- conversion_explicita: define las reglas para realizar una conversión explícita.
- expresion: define las operaciones de suma y resta entre una expresión y un término. Se definieron con asociatividad a izquierda.
- termino: define las operaciones de multiplicación y división entre un término y un factor. Se definieron con asociatividad a izquierda.
- factor: define que un factor puede ser una constante o un identificador.
- cte: define que las constantes pueden ser enteros o enteros largos, negativos o positivos. Verifica además los rangos para las constantes positivas y agrega a la tabla de símbolos las constantes negativas, borrando las positivas cuando corresponde.

Lista de errores léxicos y sintácticos

Errores lexicos

- En la acción semántica 2 se consideró el mensaje de advertencia “Identificador truncado”.
- En la acción semántica 3 y 4 se consideró el mensaje de advertencia “Constante fuera de rango, se le asignará el mayor valor permitido”.
- En la acción semántica 5 consideramos el mensaje de error “Palabra reservada no válida”.
- En la acción semántica 6 consideramos el error de “Caracter invalido”.

Errores Sintácticos

- En la regla función se consideran los errores correspondientes al tipo de función incorrecto. También se considera la falta del identificador, los “()”, las “{}”, la palabra reservada return, el retorno y el cuerpo de la función.
- En la regla declaracion_variables se consideran los errores de falta del tipo, identificador o lista de identificadores, y la “,” final.
- En la regla sentencia_seleccion se consideran los errores de falta de la palabra reservada if, then y end_if, como así también la falta de la “,”.
- En la regla condicion_if se consideran los errores de la falta de “()”.
- En la regla sentencia_control se consideran los errores correspondientes a la falta de las palabras reservadas “case” y “do”, también la falta de “()” y “{}, y ademas la falta del identificador, opciones o “,”
- En la regla sentencia_impresion los errores correspondientes a la falta de la palabra reservada “print”, la cadena de caracteres, los “()” o la “,”.
- En la regla sentencia_asignacion los errores correspondientes a la falta del identificador, el símbolo de asignación, la expresión o función del lado derecho y la falta de la “,”.
- En la regla condicion los errores de falta de expresiones de uno o ambos lados, como así también del operador de comparación.

Consideraciones

-Como el código puede contener valores de enteros y enteros largos negativos se optó por usar un atributo contador de referencias en los token 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. Por eso mismo se utiliza dicho contador, borrando las constantes que se ingresaron de manera incorrecta a la tabla de símbolos.
-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.