Universidad Nacional del Centro de la Provincia de Buenos Aires.

Trabajo Práctico

Diseño de Compiladores

Re-entrega Analizador Léxico y Sintáctico

Grupo 8

INTEGRANTES:	CORREO:
Chiozza, Juan Ignacio	chiozzajuani@gmail.com
González, Cristian	cris_mdq22@hotmail.com
Rodríguez, Facundo Hernán	farodriguez@alumnos.exa.unicen.edu.ar

Contenido

1. INTRODUCCION

A lo largo del presente trabajo se busca crear un compilador para un lenguaje diseñado con el objetivo de conocer y aprender el funcionamiento de los diferentes compiladores que utilizamos. Para ello, se decidió implementarlo en el lenguaje Java y se utilizó el programa “YACC” para generar el analizador léxico.

El lenguaje que soporta nuestro compilador tiene las siguientes funcionalidades especiales:

- Doubles: Números reales con signo y parte exponencial.
- Estructura de iteración DO – UNTIL.
- Comentarios de una línea.
- Cadenas multilíneas.
- Colecciones con métodos predefinidos y rowing.

2. DECISIONES DE DISEÑO E IMPLEMENTACION

Como primer y principal paso se implementó una clase principal *"AnalizadorLexico"* encargada de obtener y distribuir la distinta información a analizar a quien le corresponda. De esta forma, se distribuyeron las distintas etapas del análisis en distintas clases. Con el fin de evitar la lectura de espacios, saltos de línea y reconocer el fin del archivo se implementó la clase "*ControladorArchivo*" el cual a partir del texto completo del código entrega el próximo carácter a leer cada vez que se le solicite y, a su vez, permite retroceder en la lectura en caso de ser necesario.

Por otra parte, se creó la clase abstracta *"Accion_Semantica"* encargada de ejecutar cada una de las acciones semánticas asociadas a cada estado, almacenando información en la tabla de símbolos, informando acerca de un error o una advertencia (warnings) y hasta realizando un truncamiento en el valor de una constante en caso de ser necesario. Para informar acerca de los errores y warnings encontrados durante la compilación se creó la clase Error, encargada de almacenar la información necesaria para detallar lo ocurrido.

Para guardar la información referida a los tokens encontrados se creó la clase homónima, guardando dicha información de manera dinámica en una HashTable con sus atributos, aunque se forzó a la clase (mediante el constructor) a contener los atributos "Tipo", "Número de Token" y "Valor".

Para implementar la tabla de símbolos se utilizó una hash ya que se requiere una estructura dinámica en donde la clave es el valor de la constante o cadena y el nombre en los identificadores.

3. DIAGRAMA DE TRANSICION DE ESTADOS

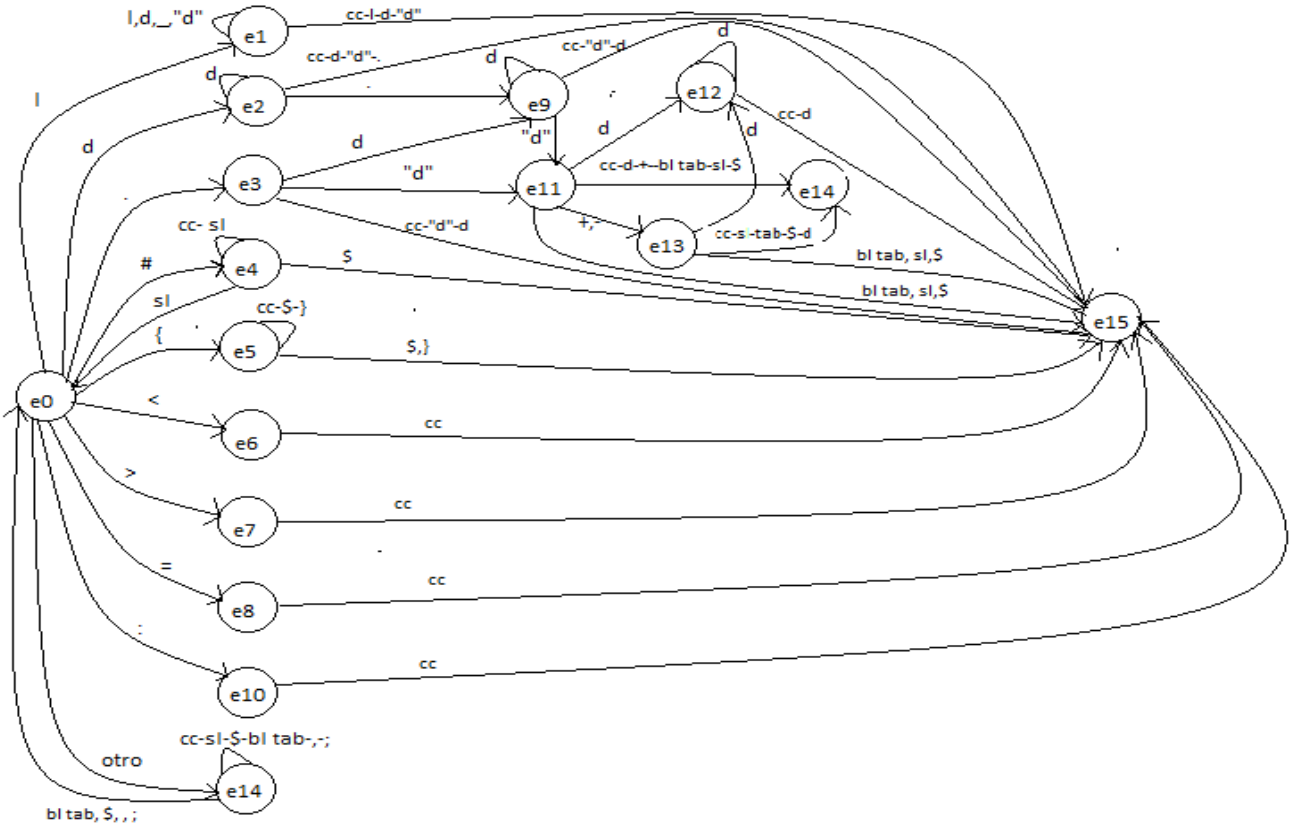

Figura 2: Diagrama de Transición de Estados[1].

REFERENCIAS:

- *d: Dígito*
- *l: Letra (Minúscula o Mayúscula)*
- *"d": Letra d o D.*
- *Sl: Salto de Línea*
- *cc: Cualquier caracter.*
- *$: fin de archivo.*
- *E15: Estado Final*
- *E0: Estado Inicial.*
-

[1] El estado 14 se repite en el diagrama por simplicidad en el diagrama.

4. MATRIZ DE TRANSICION DE ESTADOS

Estado	Letra	Díg	/	*	+	-	=	<	>	#	(	)	}	,	;	_	{	D/d	.	[	]	:	Otro	Bl tab	SL	$
0	1	2	F	F	F	F	8	6	7	4	F	F	F	F	F	F	5	1	3	F	F	10	14	F	F	F
1	1	1	F	F	F	F	F	F	F	F	F	F	F	F	F	1	F	1	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	14	9	F	F	F	F	F	F	F
3	F	9	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	11	F	F	F	F	F	F	F	F
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	0	F
5	5	5	5	5	5	5	5	5	5	5	5	5	F	5	5	5	5	5	5	5	5	5	5	5	5	F
6	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
9	F	9	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	11	F	F	F	F	F	F	F	F
10	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
11	14	12	14	14	13	13	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	F	F	F
12	F	12	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
13	14	12	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	F	F	F
14	14	14	14	14	14	14	14	14	14	14	14	14	14	0	0	14	14	14	14	14	14	14	14	0	0	0

Figura 3. Matriz de transición de estados

5. ACCIONES SEMÁNTICAS ASOCIADAS

Para implementar las diferentes acciones semánticas se buscaron diferentes estados que compartían las mismas acciones a realizar.

El primer grupo (Acción Semántica Agregar Buffer) son aquellas transiciones en los que no se llega a un estado final. Para este grupo la acción semántica realiza únicamente la inserción del carácter que está leyendo actualmente al buffer temporal.

El segundo grupo (Acción Semántica Error), son aquellas transiciones que nos llevan a un estado de error de Compilación, por lo que se detecta el tipo de error para luego informarlo.

El tercer grupo (Acción Semántica Operador Final) representa a los operadores identificados como token. En este caso, solamente devolverá el carácter ASCII correspondiente al operador identificado. Cabe aclarar que, en caso de un error léxico, el analizador indica a la acción semántica mediante el constructor de esta, que debe retroceder en el archivo y retornar el operador identificado.

Análogamente se crearon distintas acciones semánticas para los identificadores, cadenas multilíneas, constantes doubles y constantes enteras (Acción Semántica Identificador Final, Acción Semántica Double Final, Acción Semántica Enteros Final y Acción Semántica Cadena Final). En estos casos, a diferencia de los anteriores grupos, además de retornar el token correspondiente, se inserta el token y su contenido en la tabla de símbolos.

Además, en las acciones semánticas relacionadas con las constantes enteras, identificadores y constantes de tipo double, se verificó que el valor del token cumpla con las condiciones propuestas por el analizador léxico:

- Constantes enteras y doubles dentro del rango permitido,
- Identificadores cuyos nombres tengan no más de 25 caracteres.

En el caso particular de los identificadores, se identificó y diferenció los identificadores de las palabras reservadas.

5.1 MATRIZ DE ACCIONES SEMANTICAS

Estado	Letra	Díg	/	*	+	-	=	<	>	#	(	)	}	,	;	_	{	D/d	.	[	]	:	Otro	Bl tab	SL	$
0	AB	AB	OF	OF	OF	OF	AB	AB	AB	-	OF	OF	OF	OF	OF	OF	-	AB	AB	OF	OF	AB	E1	-	-	-
1	AB	AB	IF	IF	IF	IF	IF	IF	IF	IF	IF	IF	IF	IF	IF	AB	IF	AB	IF	IF	IF	IF	IF	IF	IF	IF
2	EF	AB	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	E2	AB	EF	EF	EF	EF	EF	EF	EF
3	OF	AB	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF
4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	CF	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	-	CF
6	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF
7	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF
8	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF
9	DF	AB	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	AB	DF	DF	DF	DF	DF	DF	DF	DF
10	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF
11	E2	AB	E2	E2	AB	AB	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2
12	DF	AB	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF	DF
13	E2	AB	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figura 4. Matriz de Acciones Semánticas[2]

REFERENCIAS:

- OF: Acción Semántica Operador Final.
- DF: Acción Semántica Double Final.
- E1 y E2: Errores de Compilación.
- EF: Acción Semántica Entero Final.
- AB: Acción Semántica Agregar Buffer.
- IF: Acción Semántica Identificador Final.

[2] Las acciones semánticas son explicadas en el apartado 5.

6. ERRORES LÉXICOS CONSIDERADOS

Los errores léxicos considerados se muestran tanto en la matriz de transición de estados como en la de acciones semánticas. Los detalles de cada uno de ellos son los siguientes:

- *E1: "Carácter no reconocido*" : Es generado cuando no se reconoce el carácter inicial.
- *E2: "Declaración incorrecta de tipo Double" :* Es generado cuando se define incorrectamente una constante de tipo double.

Es importante destacar la forma elegida para la recuperación de errores léxicos de nuestro compilador. La estrategia elegida fue que en el mismo instante que se llega a un estado de error descartar los caracteres encontrados hasta ese momento y continuar con el siguiente caracter sin importar cual fuera. Otra alternativa, era descartar los siguientes caracteres hasta encontrar un espacio, pero, si bien en muchos casos resulta más intuitivo realizarlo de esta forma no se lo hizo porque se llega al riesgo de descartar demasiado código.

7. ANALIZADOR SINTACTICO

7.1 PROCESO DE DESARROLLO

Para implementar el analizador sintáctico se utilizó la herramienta Yacc. En una primera instancia, se implementaron las reglas para que soporte el lenguaje correctamente. Al finalizar esta etapa, se comenzaron a agregar nuevas reglas para contemplar ciertos errores que creíamos que eran los más comunes que pueden surgir.

Los mayores inconvenientes surgieron en el momento de definir las reglas para aceptar errores y que no se interrumpa el proceso de compilación. Principalmente, se analizó con mucho cuidado el uso de la regla error que provee Yacc. Generalmente, se utilizó para resolver la falta del carácter ';'.

7.2 ELEMENTOS NO TERMINALES

Para definir la gramática que soporte el lenguaje dado se implementaron un conjunto de elementos no terminales.

- *programa_principal*
- *declaraciones*
- *bloques_sentencias*

Los elementos no terminales recién nombrados son aquellos que permiten generar el cuerpo del programa. La regla programa_principal es la definida como "start" y básicamente define cómo debe estar formado el programa.

- *tipo*
- *lista variables*
- *variable*
- *declaraciones*
- *Sentencia*

El elemento no terminal "declaraciones" permite definir la declaración de variables y se utiliza para poder generar más de una de forma recursiva. Con el tipo definimos aquellos terminales que pueden ser utilizados para declarar variables, en nuestro caso int y double.

- *print*
- *sentencia_seleccion*
- *asignacion*
- *sentencia_control*

Forman el cuerpo de sentencias del programa. Nuestro lenguaje acepta sentencias del estilo print, sentencia_seleccion, asignacion y sentencia_control.

- *condicion*
- *operador*

La regla condición determina el formato de una comparación, ya sea por los operadores mayor, menor, menor igual, mayor igual o distinto. Este no terminal es utilizado por las sentencia_seleccion y sentencia_control para llevarse a cabo.

- *cuerpo_if.*
- *cuerpo_else.*

Estas reglas forman parte de las sentencias de selección. Ambas están compuestas de la misma forma: Bloques de sentencias encerradas entre un BEGIN y un END, o bien, una única sentencia. La diferencia entre ellas radica en su utilización en la regla sentencia_seleccion: sólo es posible un cuerpo_else si existe un cuerpo_if anteriormente.

- *lado_izquierdo*
- *expresion*
- *término*
- *factor*

La reglas "lado izquierdo" y "expresión" son utilizados como no terminales por la regla "asignación" mencionada anteriormente. La finalidad del lado izquierdo es limitar los terminales que pueden aparecer en ese lugar. La expresión, por su parte, utiliza término y factor para lograr la precedencia de operadores aritméticos.

- *metodos*
- *coleccion*
- *nombre_metodo*

Por último, estos no terminales tienen el fin de detectar las colecciones y sus métodos correspondientes. La regla "métodos" está compuesta por un identificador que representa a la colección, y la regla "nombre_metodo" que contiene a los nombres de los métodos disponibles (First, Last y Length).

7.3 ERRORES SINTACTICOS DETECTADOS

Un aspecto importante a la hora de implementar el analizador sintáctico fue la definición de los errores. A continuación, se mostrarán los errores considerados:

- *errorPrint1*: El error se produce cuando la cadena a imprimir no es válida.
- *errorPrint2*: Se produce cuando no se utiliza correctamente la palabra reservada PRINT.
- *errorParentesisA*: Paréntesis no equilibrados. En este caso se produce cuando no se abrieron los paréntesis.
- *errorParentesisC*: Ídem anterior, con la diferencia que en este caso el error se produce cuando no se cerraron los paréntesis.
- *errorCondicionI*: Ocurre cuando no se reconoce el lado izquierdo de la condición.
- *errorCondicionD*: Ídem anterior, pero cuando no se reconoce el lado derecho de la condición.
- *errorPuntoComa*: Se espera un ';' al final de la sentencia.
- *errorBeginIF* : Se esperaba la palabra reservada BEGIN al comienzo.
- *errorDeclaracionVar*: No se declaró correctamente las variables.
- *errorSentencias*: No se declararon correctamente las sentencias dentro del programa.
- *errorAsignacion*: Hubo un error en la asignación, ya sea porque no se reconoció el lado izquierdo o el derecho.
- *errorAsignacionIgual*: Este error se produce cuando se declara incorrectamente el operador asignación. Se esperaba el operador := pero se obtuvo =.
- *errorPalabraIF*: Ocurre cuando se encuentra la palabra if en minúsculas, dentro del cuerpo de sentencias de selección.
- *errorPalabraElse*: Ídem anterior, pero con la palabra reservada else.
- *errorDoCondicion*: Se produce cuando no existe una condición de control en la sentencia DO-UNTIL.
- *errorDOUNTIL*: Ocurre cuando no se coloca la sentencia UNTIL en el cuerpo de la sentencia de control.
- *errorEND*: Se produce cuando no existe la palabra reservada END dentro de un bloque de sentencias en un cuerpo_if o cuerpo_else.
- *errorMetodo*: El error ocurre cuando se llama a un método no reconocido en las colecciones.
- *errorDeclaracionColeccion*: Error en la declaración de la colección.

8. REENTREGA

Se corrigieron distintos erros y consideraciones particulares que se encontraron en la primera entrega:

- Detección de distintos errores sintácticos que se agregaron en la gramática,
- Corrección del autómata y su correspondiente matriz de estados,
- Modificación de la gramática para que en una colección el tamaño de esta sea únicamente una constante entera y no un identificador,
- Se agregaron distintos casos de prueba,
- Corrección de los números de línea,
- Modificación de constructores en las acciones semánticas para que solamente reciba los parámetros adecuados.

9. CONCLUSIONES

Para llevar a cabo la primer entrega fue de gran importancia el análisis y la correcta implementación del autómata y, por consiguiente, la tabla de transición de estados ya que el correcto funcionamiento del analizador léxico depende de ambos factores.

Por otro lado, definir los errores sintácticos es una tarea muy difícil ya que existen muchos y diversos tipos de errores. Por este motivo el compilador soporta solo los errores mencionados en el informe, de otro modo abortará la compilación.