DISEÑO DE COMPILADORES

Trabajos Prácticos 1 y 2

Grupo 5

Matías Prado
matiprado92@gmail.com

Camila Robles
roblescami@gmail.com

Ayudante: Nicolás Escribal

Contenidos

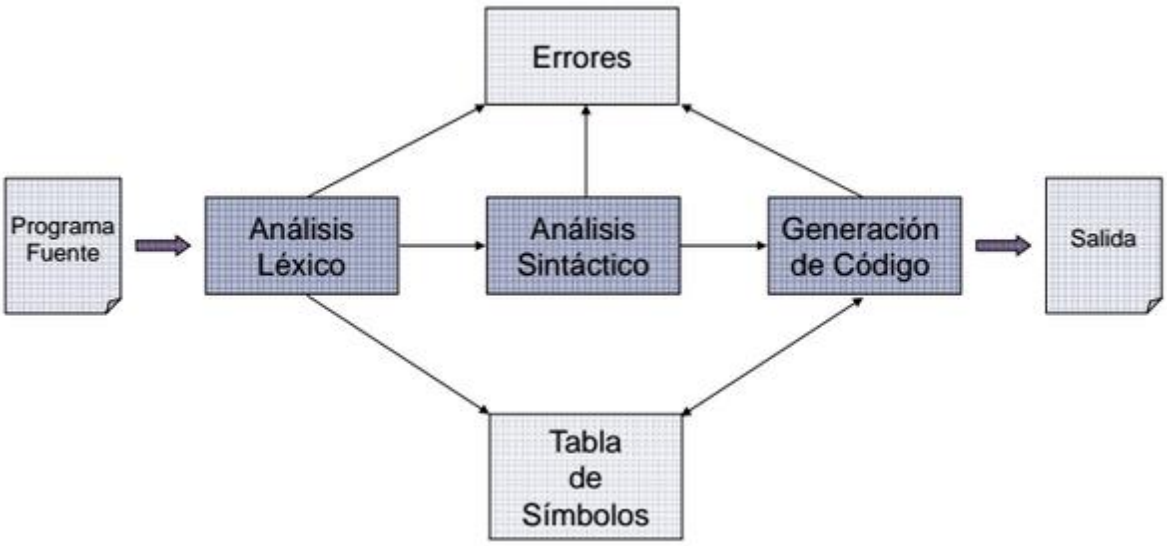
Introducción

El siguiente informe cumple la finalidad de explicar de manera detallada el proceso de elaboración de los trabajos prácticos 1 y 2 para diseño de compiladores. El objetivo es el desarrollo de un compilador capaz de leer un código fuente como entrada en un lenguaje particular, y analizarlo informando los errores encontrados en el mismo.

En esta primera entrega se desarrolló el analizador léxico y el analizador sintáctico. La estructura de un compilador consta de dos partes:

- **Analizador Léxico**: se encarga de verificar que se cumplan las reglas léxicas del código fuente dado como entrada. Su función primordial es agrupar caracteres en unidades significativas llamadas *tokens*[1]. Otras funcionalidades son: informar errores léxicos y eliminar comentarios, espacios en blanco, tabulaciones y saltos de línea.
- **Analizador Sintáctico**: verifica el cumplimiento de las reglas sintácticas. Su función es solicitar *tokens* al analizador léxico y verificar que estos respeten la gramática del lenguaje. Otra funcionalidad es informar errores sintácticos.

La aplicación desarrollada tendrá como entrada un código fuente, y realizará un análisis léxico y sintáctico para informar errores, *tokens* y estructuras sintácticas contenidas en el mismo.



[1] **Token**: es una cadena de caracteres que tiene un significado coherente en un lenguaje.

Analizador léxico

En la primera parte del trabajo se debió implementar el analizador léxico que reconozca los siguientes *tokens*:

- **ID:** con la restricción de que los nombres no pueden tener más de 15 caracteres de longitud. El primer carácter puede ser una letra o un @. En caso de que sea un @ el siguiente debe ser si o si una letra. Luego los nombres pueden ser completados con letras, @ o números, según le plazca al usuario. Solo pueden escribirse en minúscula.
- **CONSTANTES ENTERAS:** entre los valores -2^{15} y $2^{15}-1$. Estas constantes deben llevar el sufijo "_i".
- **CONSTANTES FLOTANTES:** Números reales con signo y parte exponencial. El exponente comienza con la letra E (mayúscula o minúscula) y puede tener signo. La ausencia de signo implica positivo. La parte exponencial puede estar ausente. Considerar el rango 1.17549435 e-38 < |x| < 3.40282347 e38.
- **OPERADORES ARITMÉTICOS**: "+", "-", "*", "/".
- **OPERADOR DE ASIGNACIÓN**: "=".
- **COMPARADORES:** """>=", "<=", ">", "<", "==", "<>".
- **TOKENS ESPECIALES:** "(", ")", ",", ";", " ' ", "{", "}", ".".
- **CADENAS DE CARACTERES:** cadenas de caracteres que comiencen y terminen con " ' ". Estas no pueden ocupar más de una línea.
- **COMENTARIOS MULTILINEA:** Comentarios que comiencen con "//" y terminen con "//" (pueden ocupar más de una línea).
- **PALABRAS RESERVADAS:** IF, THEN, ELSE, ENDIF, PRINT, INT, BEGIN, END, FLOAT, STRING, TOFLOAT, GLOBAL, LOOP, FROM, TO, BY.
- **FIN DE ARCHIVO.**

También debe eliminar de la entrada (reconocer pero no informar como *token*) los comentarios, el carácter en blanco, tabulación, y salto de línea.

Decisiones de diseño e implementación

El analizador léxico fue implementado en Java, siguiendo el paradigma de la programación orientada a objetos.

En principio se realizó el diagrama de transición de estados para poder ver la formación de los distintos *tokens* y luego, a partir de este, se conformó la **matriz de transición estados** y la **matriz de acciones semánticas**. Se creó una clase abstracta *SemanticAction* y trece clases concretas (SA1, SA2,..., SA13) que heredan de ella. Las clases concretas redefinen el método *execute()* debido a que cada acción semántica realiza una operación particular y diferente al resto.

Acciones semánticas

SA1: Inicializa un *token* y adiciona el símbolo leído.

SA2: Adiciona el símbolo leído al *token* recibido.

SA3: Empaqueta el *token* controlando el rango de un *float*.

SA4: Empaqueta el *token* controlando la longitud, e inicializa el vector de palabras reservadas.

SA5: Adiciona el símbolo leído y empaqueta el *token*.

SA6: Empaqueta el *token* sin adicionar el símbolo.

SA7: Lee un símbolo sin adicionarlo a ningún *token*.

SA8: Inicializa un *token*, adiciona el símbolo leído y empaqueta el *token*.

SA9: Empaqueta el *token* controlando el rango de un entero.

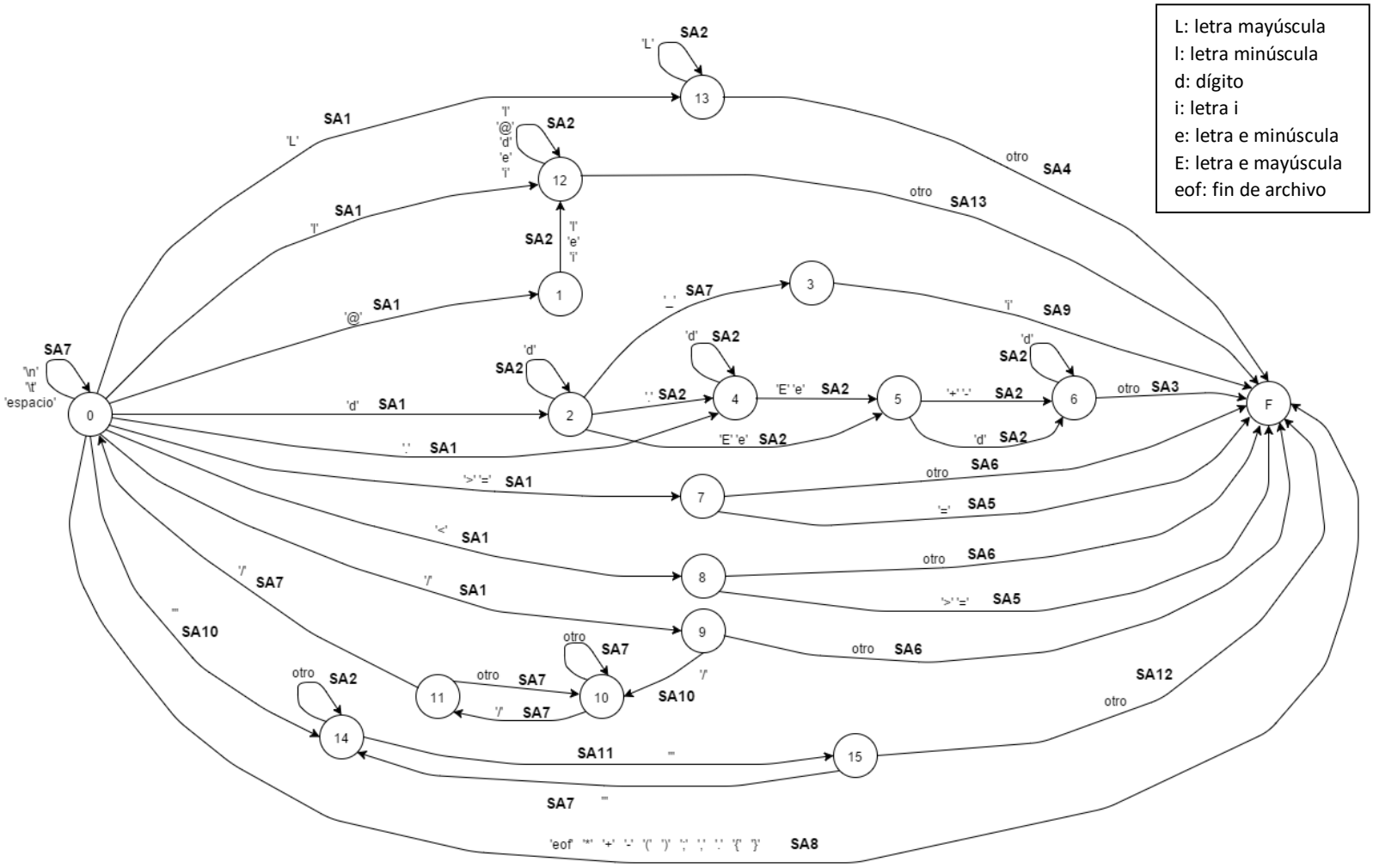
SA10: Inicializar el *token* sin adicionar el símbolo.

SA11: Vacía el *token* e informa error.

SA12: Empaqueta el *token* sin consumir el rango de un carácter.

SA13: Empaqueta el *token* controlando la longitud.

Diagrama de transición de estados



Matriz de transición de estados

	space	\n	\t	l	@	d	_	i	+	-	(	)	=	<	>	*	/	;	,	.	E	L	otro	{	}	e	EOF	´
0	0	0	0	12	1	2	-1	12	-1	-1	-1	-1	7	8	7	-1	9	-1	-1	4	13	13	-1	-1	-1	12	0	14
1	-1	-1	-1	12	-1	-1	-1	12	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	12	-1	-1
2	-1	-1	-1	-1	-1	2	3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	4	5	-1	-1	-1	-1	5	-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	-1
4	-1	-1	-1	-1	-1	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	5	-1	-1	-1	-1	5	-1	-1
5	-1	-1	-1	-1	-1	6	-1	-1	6	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	6	-1	-1	-1	-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	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-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	-1	-1	-1	-1	-1
9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	11	10	10	10	10	10	10	10	10	10	-1	10
11	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	10	10	10	10	10	10	10	10	10	-1	10
12	-1	-1	-1	12	12	12	-1	12	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	12	-1	-1
13	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	13	13	-1	-1	-1	-1	-1	-1
14	14	-1	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	-1	15
15	-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	14

Matriz de acciones semánticas

	space	\n	\t	l	@	d	_	i	+	-	(	)	=	<	>	*	/	;	,	.	E	L	otro	{	}	e	EOF	´
0	7	7	7	1	1	1	11	1	8	8	8	8	1	1	1	8	1	8	8	1	1	1	11	8	8	1	8	10
1	11	11	11	2	11	11	11	2	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	2	11	11
2	11	11	11	11	11	2	7	11	11	11	11	11	11	11	11	11	11	11	11	2	2	11	11	11	11	11	2	11	11
3	11	11	11	11	11	11	11	9	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
4	3	3	3	3	3	2	3	3	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	3	3	2	3	3
5	11	11	11	11	11	2	11	11	2	2	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
6	3	3	3	3	3	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
7	6	6	6	6	6	6	6	6	6	6	6	6	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
8	6	6	6	6	6	6	6	6	6	6	6	6	5	6	5	6	6	6	6	6	6	6	6	6	6	6	6	6
9	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	10	6	6	6	6	6	6	6	6	6	6	6
10	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
12	13	13	13	2	2	2	13	2	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	2	13	13
13	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	2	2	4	4	4	4	4	4
14	2	11	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
15	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	2

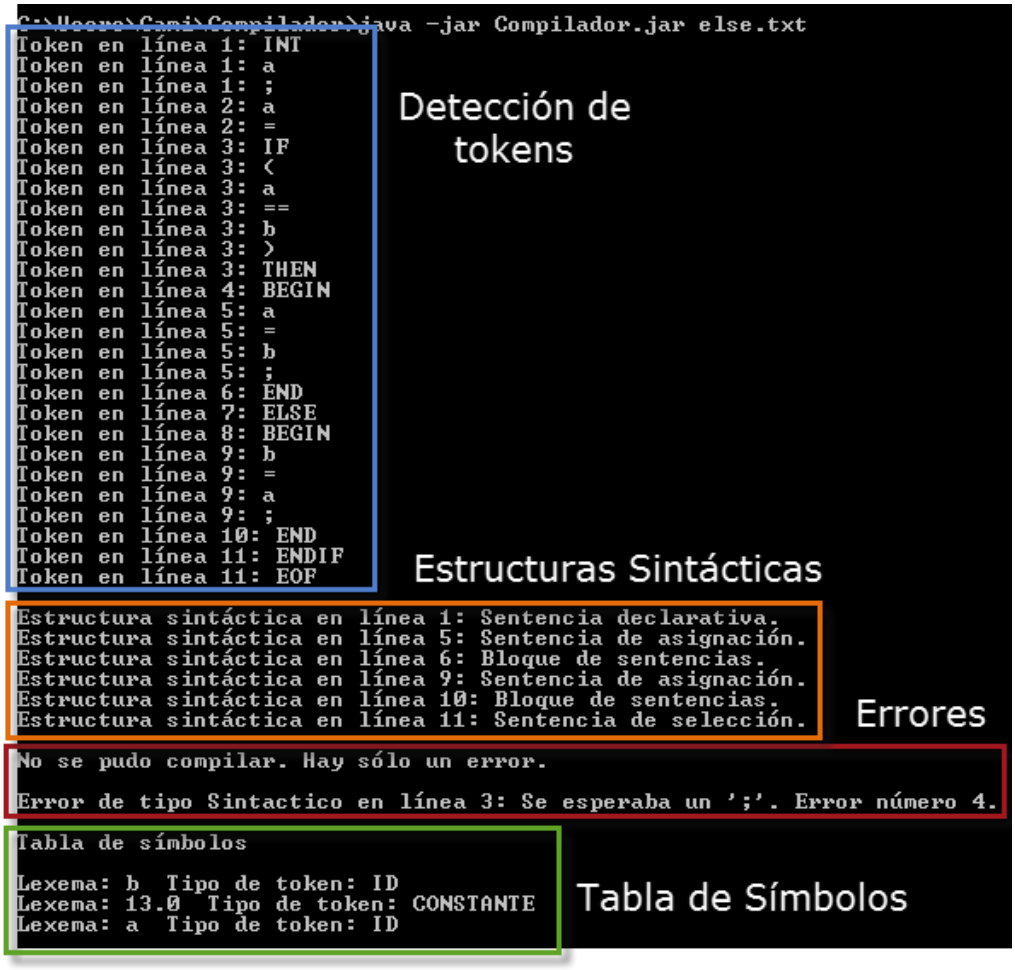
Tabla de *tokens*

Id	Nombre	Identificador numérico
ID	Identificador	265
IF	If	257
THEN	Then	258
ELSE	Else	259
BY	By	277
LOOP	Loop	263
TO	To	264
FROM	From	274
PRINT	Print	261
EOF	Fin de archivo	276
BEGIN	Begin	273
END	End	271
ENDIF	Endif	260
STRING	String	268
INT	Int	262
FLOAT	Float	267
TOFLOAT	ToFloat	275
GLOBAL	Global	272
CONSTANT	Constante Entera	266
==	Comparador	269
<=	Comparador	269
>=	Comparador	269
<>	Comparador	269
<	Comparador	269
>	Comparador	269
=	Asignación	270

En el caso de los demás caracteres se tomara como ID numérico el ASCII de cada uno.

Forma de presentación

Se decidió que el compilador muestre los mensajes de análisis de código mediante la consola.



Errores: Muestra los errores indicando la línea donde se encuentran, descripción del error, tipo de error y numero de error.

Warnings: Muestra los warnings indicando la línea donde se encuentra y una descripción.

Tokens: Muestra los *tokens* detectados junto al número de línea donde se encuentran.

Estructuras Sintácticas: Muestra las estructuras sintácticas detectadas y el número de línea donde se encuentran.

Tabla de Símbolos: Muestra una tabla donde, por cada entrada, se pueden observar los símbolos detectados y su clasificación. En esta tabla sólo se guardaran los identificadores, constantes y cadenas de caracteres.

Analizador sintáctico

Para esta instancia del trabajo práctico se implementó una gramática mediante una sintaxis similar a la de BNF[2], particularmente, la especificada para YACC[3]. Se construyó un *parser* que invoca al analizador léxico creado anteriormente, que reconoce un lenguaje que incluye:

Programa

Programa constituido por sentencias declarativas seguidas de sentencias ejecutables. El programa no tendrá ningún delimitador. Cada sentencia debe terminar con ";".

Sentencias declarativas

Sentencias de declaración de datos para los tipos de datos con la siguiente sintaxis:
<tipo> <lista_de_variables>;
Donde <tipo> puede ser: INT, FLOAT o STRING.
Las variables de la lista se separan con coma (“,”).

Sentencias ejecutables

Cláusula de selección (IF): Cada rama de la selección será un bloque de sentencias. La condición será una comparación entre expresiones aritméticas, variables o constantes, y debe escribirse entre ( ). La estructura de la selección será, entonces:
IF (<condición>) THEN <bloque_de_sentencias> ELSE <bloque_de_sentencias> ENDIF

El bloque para el ELSE puede estar ausente.

Un bloque de sentencias puede estar constituido por una sola sentencia, o un conjunto de sentencias delimitadas por BEGIN y END.

Cláusula de iteración (LOOP FROM TO): LOOP FROM i = n TO m BY j <bloque_de_sentencias>

Sentencia de salida de mensajes por pantalla: El formato será PRINT (cadena).

Condición: Los operandos de las expresiones aritméticas pueden ser variables, constantes, u otras expresiones aritméticas. No se permiten anidamientos de expresiones con paréntesis.

[2] **BNF (Backus-Naur Form):** metalenguaje usado para expresar gramáticas libres de contexto: es decir, una manera formal de describir lenguajes formales.
[3] **YACC** provee una herramienta general para analizar estructuralmente una entrada.

Ámbitos con nombre

Los ámbitos deben corresponderse con la siguiente especificación:

<nombre>
{
  Sentencias declarativas (no son obligatorias)
  Sentencias ejecutables
}
<nombre> será un identificador, que no podrá ser utilizado como variable en otro lugar del programa.

Modificación de reglas de alcance

Global: Entre las sentencias declarativas de los ámbitos, se podrán incluir una con la sintaxis:

GLOBAL <lista de variables>

Conversiones

Explícitas: En cualquier lugar donde pueda aparecer una variable o una expresión, podrá aparecer:

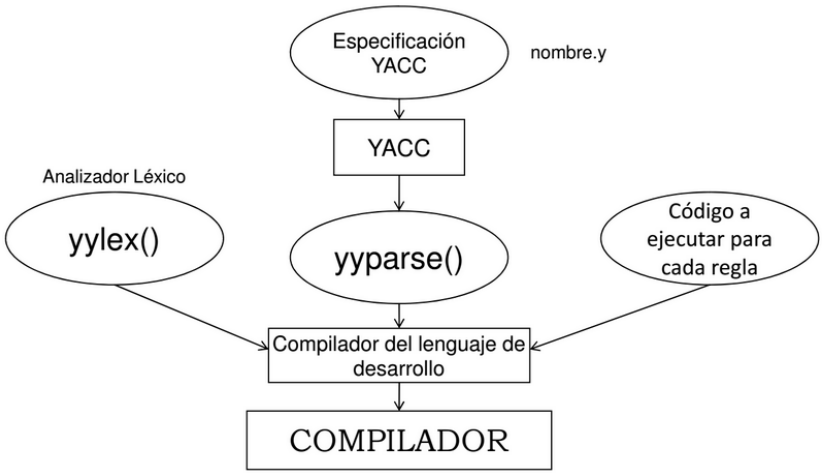
TOFLOAT (<expresión>)

Comentarios

Los comentarios comenzarán con "**//**" y terminarán con "**//**". Los comentarios podrán tener más de una línea.

Introducción a YACC

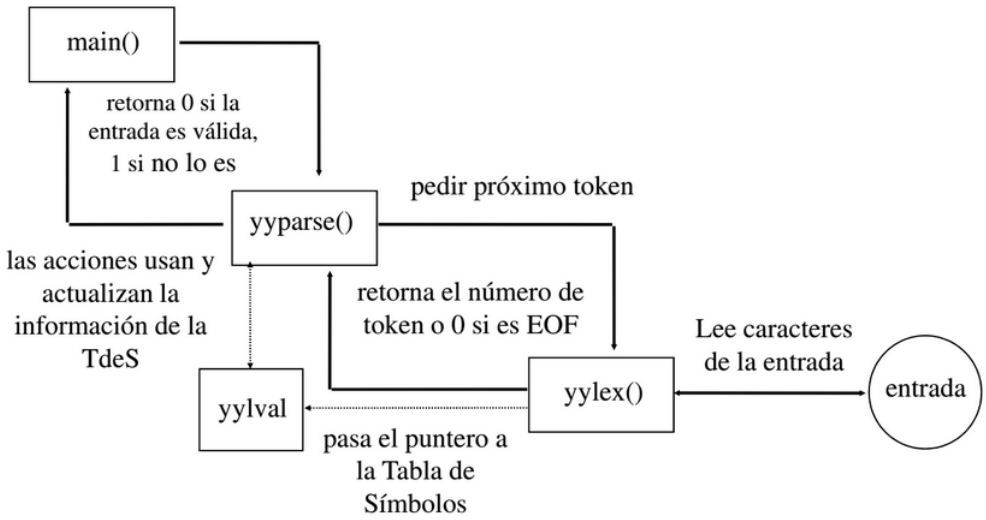
El analizador léxico es el encargado del reconocimiento de *tokens* que son procesados por el analizador sintáctico para construir un árbol de análisis. Para esto se utilizó la herramienta YACC, que se encarga de realizar el árbol de *parsing* de forma automática.



Para la utilización de YACC se debió aprender y utilizar la siguiente sintaxis:

%{
}%
DECLARACIONES
%%
REGLAS
%%
CODIGO

La sección de declaraciones es el lugar donde se pueden definir los *tokens*, precedencias, entre otros, que se van a utilizar en las reglas. En la sección de reglas se genera la gramática y las acciones semánticas correspondientes son analizadas gramaticalmente. La sección de código es opcional. Puedo poner aquí una rutina o método de manejo de errores, el *main* o el léxico. Los %% se encargan de separar las secciones.
En el siguiente diagrama se puede ver el funcionamiento de la herramienta:



Desarrollo

El desarrollo de la gramática se fue realizando en forma cautelosa, ya que de cualquier otra manera se generarían conflictos *shift/reduce* y *reduce/reduce*. Se comenzó definiendo las declaraciones y asegurando su correcto funcionamiento y manejo de errores, para luego seguir con las distintas sentencias que aceptaba el programa.
Con la definición de *yylex()* que se utilizó, la intención fue que el analizador léxico realice toda la tarea y que el analizador sintáctico solo reciba los valores del *token* y de *yylval*. Las primeras gramáticas que se definieron fueron la del programa y las declaraciones, y luego se prosiguió con las sentencias ejecutables.
No se encontraron mayores inconvenientes con la gramática de programa y declaraciones, simplemente algunos problemas de *shift/reduce* y *reduce/reduce* al intentar mostrar los errores de los ámbitos, las iteraciones y las conversiones. Al comprender la dinámica de declaración de reglas pudimos solucionar estos errores y evitar que ocurran en futuras declaraciones.

Lista de no terminales

Fueron definidos los siguientes no terminales en la construcción de la gramática:

- **p**: Raíz del programa, un programa es aceptado si se llega a esta regla
- **declarations**: Declaración de variables
- **declaration**: Especifica los distintos tipos de declaración
- **var_list**: Lista de variables separadas por comas
- **type:** Especifica el tipo
- **exe_sentences**: Lista de sentencias de ejecución
- **sentence**: Posibles sentencias ejecutables
- **asignation**: Sentencia de asignación
- **selection**: Sentencia condicional IF THEN ELSE ENDIF
- **selection_simple:** Sentencia condicional IF
- **iteration**: Sentencia de iteración LOOP FROM TO
- **ambit:** Ámbito con nombre
- **condition**: Sentencia de comparación de expresiones
- **print**: Sentencia de PRINT, impresión por pantalla
- **conversion**: Sentencia de conversión TOFLOAT
- **bloque**: Bloque de declaraciones y ejecuciones
- **expression**: Sentencias de expresión de suma y resta
- **term**: Sentencia de expresión de mayor precedencia, para división y producto
- **factor**: Constante o Id
- **ambit_declarations:** Declaración dentro de ámbitos
- **ambit_dec_sentence:** Tipos de declaraciones dentro de los ámbitos
- **exe:** Sentencias ejecutables o ámbitos
- **bloque_exe_sentences:** Sentencias ejecutables de un bloque

Lista de errores léxicos y sintácticos

Errores Sintácticos

1: "El programa finalizó con errores. Error número 1."
2: "Falta el bloque de sentencias ejecutables. Error número 2."
3: "Falta el bloque de sentencias declarativas. Error número 3."
4: "Se esperaba un ';'. Error número 4."
5: "Falta el tipo de la declaración. Error número 5."
6: "Sentencia declarativa incorrecta. Error número 6."
7: "Falta el identificador de la asignación. Error número 7."
8: "Falta el identificador de la asignación y se esperaba un ';'. Error número 8."
9: "Bloque de sentencias sin finalizar falta END. Error número 9."
10: "Bloque de sentencias sin inicializar falta BEGIN. Error número 10."
11: "Falta abrir paréntesis '('. Error número 11."
12: "Falta cerrar paréntesis ')'. Error número 12."
13: "Parámetro del imprimir incorrecto. Error número 13."
14: "Falta palabra reservada 'PRINT'. Error número 14."
15: "Sentencia incorrecta. Error número 15."
16: "Ámbito sin finalizar falta '}'. Error número 16."
17: "Falta nombre del ámbito. Error número 17."
18: "Falta LOOP en iteración. Error número 18."
19: "Falta FROM en iteración. Error número 19."
20: "Falta TO en iteración. Error número 20."
21: "Falta BY en iteración. Error número 21."
22: "Falta una variable en iteración. Error número 22."
23: "Falta expresión. Error número 23."
24: "Falta comparador. Error número 24."

Errores Léxicos

1: "Float fuera de rango. Error número 101."
2: "Carácter no identificado. Error número 102."
3: "Construcción de token erróneo. Error número 103."
4: "Constante entero fuera de rango permitido. Error número 104."

Conclusión

La realización del trabajó sirvo para finalizar la comprensión de los temas vistos hasta ahora en la materia y sus correlativas. Adquirimos la experiencia de haber realizado de forma práctica los contenidos teóricos que involucran la implementación de tanto un analizador léxico como el de uno semántico junto a su gramática.