Universidad Nacional Autónoma de México

Facultad de Ingeniería

Práctica 3

Análisis Léxico

ALUMNOS:

CÁRDENAS CÁRDENAS JORGE
MURRIETA VILLEGAS ALFONSO
REZA CHAVARRIA SERGIO GABRIEL
VALDESPINO MENDIETA JOAQUIN

PROFESOR:

ADRIAN ULISES MERCADO MARTINEZ

1. Objetivo

Elaborar el analizador léxico como un módulo que pueda ser conectado posteriormente con el resto del compilador.

2. Desarrollo

Para la siguiente gramática:

1. program → declaracion **\n** funciones
2. declaraciones → tipo lista_var **\n** | **registro \n inicio** declaraciones **\n fin \n** declaraciones | ϵ
3. tipo → base tipo_arreglo
4. base → **ent** | **real** | **dreal** | **car** | **sin**
5. tipo_arreglo → [**num**] tipo_arreglo | ϵ
6. lista_var → lista_var **,** **id** | **id**
7. sentencias → sentencias **\n** sentencia | sentencia
8. sentencia → **si** expresion_boolena **entonces \n** sentencias **\n fin**
   | **si** expresion_boolena **sino \n** sentencias **\n fin**
   | **mientras** expresion_boolena **hacer \n** sentencias **\n fin**
   | **hacer \n** sentencia **\n minetras que** expresion_booleana
   | **id :=** expresion | **escribir** expresion | **leer** variable | **devolver** | **devolver** expresion |
9. expresion_booleana → expresion_booleana **oo** expresion_booleana | expresion_booleana **yy** expresion_booleana | **no** expresion_booleana | relacional | **verdadero** | **falso**
10. relacional → relacional < relacional | relacional > relacional | relacional <= relacional | relacional >= relacional | relacional == relacional | relacional <> relacional | expresion
11. expresion → expresion + expresion | expresion − expresion | expresion ∗ expresion | expresion / expresion | expresion % expresion | (expresion) | variable | **num** | **cadena** | **caracter** | **id**(parametros)
12. param_arr → **id**[] | param_arr []
13. variable → **id** parte_arreglo | **id.id**
14. parte_arreglo → [expresion] parte_arreglo | ϵ
15. parametros → lista_param | ϵ
16. lista_param → lista_param , param | param
17. param → **id** | param_arr
18. funciones → **func** tipo **id(** argumentos **) inicio \n** sentencias **\n fin \n** funciones | ϵ
19. argumentos → lista_arg | **sin**
20. lista_arg → lista_arg arg — arg
21. arg → tipo **id**

(a) **Separación de Terminales y No Terminales**

$\Sigma = \{$

\n	**mientras**	>=
registro	**hacer**	==
inicio	**mientras que**	+
fin	**:=**	−
ent	**escribir**	*
real	**leer**	/
dreal	**devolver**	%
car	**oo**	(
sin	**yy**	)
num	**no**	**cadena**
,	**verdadero**	**caracter**
id	**falso**	[
si	<	]
entonces	>	**.**
sino	<=	**func**

}

$N = \{$

program	**sentencias**	**parte_arreglo**
declaraciones	**sentencia**	**lista_param**
tipo	**expresion_booleana**	**param**
lista_var	**expresion**	**funciones**
base	**variable**	**argumentos**
tipo_arreglo	**parametros**	**lista_arg**
lista_var	**param_arr**	**arg**

}

3. Diseño de la solución

3.1. Expresiones regulares

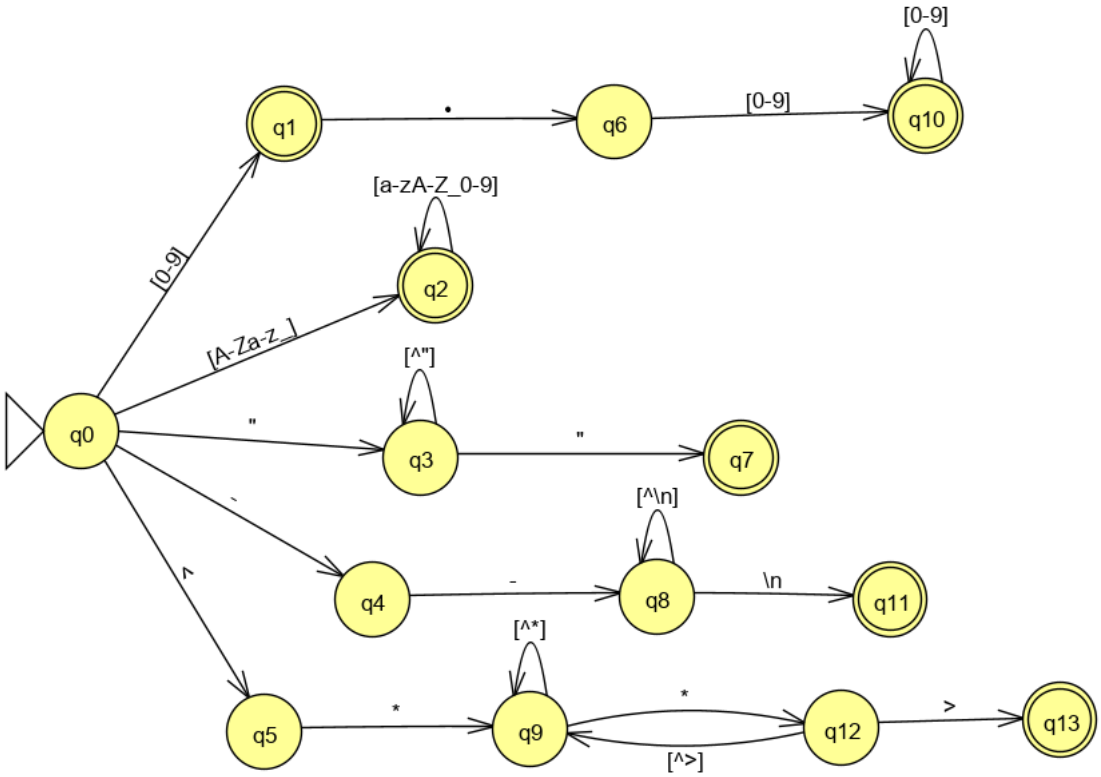
A continuación, se muestran las expresiones regulares empleadas en el código:

- entero→ $[0-9]+$
- real→entero . entero
- num→entero — real
- id→[_a-zA-Z][_0-9a-zA-Z]*
- cad→ "[^¨]*"
- car→ '.'
- ComentarioL → $--[\hat{}\text{- } \backslash n]^*\backslash n$

- cb $\rightarrow$ [ˆ<> ∗] ∗ [” ”\n\t]∗
- ComentarioM $\rightarrow$ ” < ∗” *cb* ∗ ”∗ > ”

3.2. AFD

A continuación se muestra el autómata obtenido para esta práctica:



4. Código

4.1. Organización del Código

El código analizadorM.l está dividido en 3 grandes secciones (Debido al formato que se pide Lex):

1) Sección de declaraciones generales:

   En esta parte solamente se llamó a la biblioteca stdio.h, y se definieron por un lado 2 variables globales 'estadoA' que es la encargada de verificar el estado de los comentarios (Sí están o no cerrados) y 'cc' que es una variable encargada de contar caracteres al momento de revisar cada token.
   Por último, también se definieron las expresiones regulares previamente mencionadas en el apartado de 3.1.

2) Sección de expresiones regulares:

   En este apartado se le asignó a cada token un número con el cual posteriormente se les identificará, además cabe destacar que mediante la variable 'cc' y con ayuda de yyleng es como se pudo obtener la dimensión de

cada token (La cantidad de caracteres), esto debido a que para el manejo de errores se pueda de manera un poco más precisa identificar en que parte se encuentra un error.

Cabe destacar que de manera general hay 3 grandes grupos en esta sección , el primero son los de tipo palabra reservadas, también el de operadores y caracteres especiales y por último, los encargados de identificar comentarios.

3) Sección de código en C

En esta sección solamente se declararon 2 funciones, la función 'main()' y la función 'tokens()'. La descripción de cada una de estas funciones se encuentra a detalle en el código fuente, de manera general, la función main simplemente es la encargada de abrir y cerrar los archivos con los que se trabajará, también es la encargada de utilizar las variables 'yyin' y 'yyout' y por último, llamar a la función tokens().
Por otro lado, la función 'tokens()' es la encargada de escribir tanto en consola como en en el archivo de salida cada uno de los tokens del archivo de entrada. (Para más detalles, checar la documentación en el código).

4.2. Formato de salida

Para el formato del archivo de salida se consideraron los siguientes dos casos:

- Caso 1
  $< clase_lexica, token_capturado >$
- Caso 2
  $< token_capturado >$

Tanto en consola y en el archivo de salida llamado 'salida.txt' se dará la impresión de los casos obtenidos en el analizador

5. Conclusiones

5.1. Cárdenas Cárdenas Jorge

La presente practica nos permitió generar un analizador léxico en Flex, para dicho fin fue necesario aplicar la teoría de los distintos tópicos vistos en clase, tal como el manejo de expresiones regulares y creación de autómatas finitos deterministicos a partir de estas, así como algunas funciones propias de Flex.

Es importante destacar la gran utilidad que tiene Flex para realizar el análisis léxico, ya que de manera casi automática genera los tokens asociados a cada símbolo terminal de la gramática lo que facilita en gran medida la tarea de programación de esta esta etapa de análisis.

5.2. Murrieta Villegas Alfonso

En la presente práctica se realizó con ayuda de Lex un analizador Léxico con base a todos los conocimientos previos como son el manejo de Expresiones Regulares, los ejercicios y manejo de variables y funciones propias Lex y el manejo de AFD's.
Así y con ayuda de Flex, es como se pudo identificar token a token de un archivo de entrada, esto a través del uso de Expresiones Regulares. Además,también se logró hacer de manera muy general un manejador de errores léxicos.
Por último, se pudo retornar los tokens tanto en consola como en un archivo de salida para posteriormente utilizarlos en la siguiente parte de nuestro compilador.

5.3. Reza Chavarria Sergio Gabriel

En conclusión se pudo realizar un analizador léxico con los conceptos y conocimientos teóricas vistas en clase. Esto siendo tomado en cuenta por el uso de las expresiones regulares para el reconocimiento de tokens de un archivo y en el uso de la herramienta de Lex para la identificación de estos por medio de código.

5.4. Valdespino Mendieta Joaquín

En conclusión se pudo elaborar el analizador léxico con las características especificadas, haciendo uso de Flex para la identificación de cada token en un archivo mediante el diseño de las expresiones regulares y su jerarquía en código, aplicando los conocimientos obtenidos de las materias de compiladores y programación. Por otro lado se logra observar la parte del modulo la cual se utilizará para conectar con el resto del compilador, que es la sección de retorno de tokens que además de guardarlos en un archivo devuelve un numero asociado al token.

6. Referencias

- Sergio Gálvez Rojas, Miguel Ángel Mora Mata. Compiladores. Traductores y Compiladores. Dpto. de Lenguajes y Ciencias de la Computación. Universidad de Málaga.
- LEX El analizador Léxico. Dpto. de Lenguajes y Ciencias de la Computación. Universidad de Magala
- Federico Simmross Wattenberg. El generador de analizadores léxicos LEX. Universidad de Valladolid.