UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Compiladores

Analizador Léxico

Presentan

German López Rodrigo
Lemuz Fuentes Omar Alejandro

Fecha Entrega:
25 / 09 / 2018

Grupo:
04

PROFESORA

Sandoval Montaño Laura

ÍNDICE

Descripción del Problema

¿Cuántas veces, después de programar, hemos compilado el programa sin darnos a la tarea de saber que conlleva este proceso? Sin duda, los compiladores son herramientas con las que tenemos contacto todos los días como ingenieros en computación en proceso, sin embargo, su funcionamiento es algo que muchas veces desconocemos; nos limitamos a asumir que la compilación es un paso indispensable cuando programamos, pero no más. Por esta razón es que, a través del programa desarrollado en este trabajo, pretendemos explicar y entender cómo es que funciona una de las partes de los compiladores: el análisis léxico.
Como ya sabemos, el análisis léxico reconoce los componentes léxicos de un programa, y aunque esto suene un tanto redundante, es importante que se mencione. Asimismo, el analizador léxico posee funciones más específicas; ya que se encarga de:

1. Crear tablas estáticas y dinámicas para los componentes léxicos.
2. Genera tokens (uno por cada palabra reconocida).
3. Lee el programa fuente.
4. Detecta errores léxicos.
5. Detecta y descarta comentarios.
6. Detecta y descarta espacios, saltos de línea y tabuladores.

El analizador léxico, al ser el primero en ejecutarse una vez que inicia el proceso de compilación, es importante que sea construido de manera correcta para que no existan errores futuros en los demás componentes del compilador.

Propuesta de Solución

- ### Análisis

  Para poder llevar a cabo este trabajo, la planificación usada fue la división de tareas y la complementación de estas mediante el trabajo en equipo, es decir, aunque uno de nosotros se enfocara en realizar una parte del programa en específico, existía comunicación en cuanto a las dudas que surgieran e incluso en ocasiones era posible que, si la modificación de esa parte era necesaria, la otra persona que no estaba involucrada específicamente en esa actividad modificara libremente el programa para el beneficio de ambos.

- ### Diseño e Implementación

  Para poder cumplir con el objetivo propuesto en el presente, se hará uso de los lenguajes C y Flex en conjunto. Flex nos facilita el proceso de construcción del programa gracias a las funciones que posee (enfocadas en la definición de lenguajes regulares a través de expresiones).

  De esta manera, se construyó un conjunto de expresiones regulares para poder cumplir con los requerimientos necesarios para la construcción del lenguaje planteado. Dichas expresiones en este caso específico, para cada una de las clases, fueron:

  - dig [0-9] (D
  - letrasAny [a-zA-Z]
  - coment \$\$.*\n
  - palRes Bul|Cadena|Cierto|Entero|Falso|Haz|Mientras|Para|Real|Si|Sino
  - ident [a-z]{letrasAny}{1,7}
  - simbEsp [\(|\)|\,|\[|\]|\;]
  - opAsig ([:|+|\-|*|\/|&|\^|!|%]|[<<|>>]{2,2})=
  - opRel \.DIF\.|\.IGL\.|\.MN\.|\.MNI\.|\.MY.|\.MYI\.
  - opArit [+|\-|*|\/|%|=]
  - cteNumEntera {dig}+
  - cteNumFloat {dig}*\.{dig}+
  - cteNumFloat2 {dig}+\.{dig}*

- cteNumScie ({cteNumFloat}|{cteNumFloat}|{cteNumEntera})[E|e][+|\-]?{dig}+
- cteNumReal ({cteNumFloat}|{cteNumFloat2}|{cteNumScie})
- cteCadena \x22.*\x22
- salto \n
- space \t|" "
- error .

Las clases a las cuales satisfacen las expresiones anteriores son:

Clase	Descripción
0	Palabras reservadas
1	Identificadores (sólo letras, inicia con minúscula y hasta 8 letras)
2	Símbolos especiales ( ) , ; [ ]
3	Operador de asignación
4	Operadores relacionales
5	Operadores aritméticos + - * / %
6	Constante cadena (entre comillas)
7	Constantes numéricas enteras (base 10)
8	Constantes numéricas reales Ejemplo: 2.17, .39, 437. 2.7E-3, 57E5, 18e+5

De esta manera, con las expresiones regulares definidas y algunas funciones auxiliares en lenguaje C, fue construido el analizador léxico.

Además de la tabla de clases se construyeron distintas tablas por cada una de las clases; estas tablas podían ser estáticas o dinámicas.

Las tablas estáticas fueron determinadas en las especificaciones de este trabajo. Éstas fueron:

- Tabla de palabras reservadas.

Valor	Palabra reservada
0	Bul
1	Cadena
2	Cierto
3	Entero
4	Falso
5	Haz
6	Mientras
7	Para
8	Real
9	Si
10	Sino

- Tabla de operadores relacionales.

Valor	Operador relacional
0	.DIF.
1	.IGL.
2	.MN.
3	.MNI.
4	.MY.
5	.MYI.

Las tablas estáticas reciben este nombre ya que los valores que pueden brindar a los tokens generados por este componente del compilador son fijos.

Debido a que muchos de los valores en tablas para el caso de otras expresiones regulares no se pueden definir de manera estática (operadores de asignación, cadenas, etc.), se implementaron estructuras de datos para poder definir las tablas correspondientes a los elementos que componen la clase para las expresiones mencionadas de forma dinámica, es decir, este tipo de tablas será distinto para cada uno de los archivos de entrada que reciba el analizador como entrada; el valor que

reciba cada elemento dependerá del momento en el que éste se agrega a la tabla correspondiente.

Otra de las herramientas que fue utilizada para poder llevar a cabo este trabajo fue el manejo de archivos. Dicha herramienta fue crucial para poder concluir esta parte del compilador, ya que tanto en la entrada como en la salida se requiere del uso de archivos para recibir y posteriormente brindar información útil para la siguiente fase de compilación.

Mediante la declaración de la variable “FILE *archivo”, y de las funciones “fopen”, “fprintf” y “fclose” es posible realizar un manejo básico y sencillo, pero necesario, de los archivos de entrada y salida.

Tambien fue necesario utilizar Tecnicas de busqueda y de inserción para poder llevar a cabo este trabajo, debido a que sin ellas no era posible el manejo y la creación de la tablas dinámicas. Las técnicas de búsqueda y de inserción que se emplearon fueron las más simples debido a que no podíamos alterar el orden de las tablas ya que si las alterábamos provocaríamos incongruencias en la salida del Analizador Léxico.

Instrucciones

Para poder correr el programa es necesario tener instalado flex en nuestro equipo, en caso de contar con éste lo único que se debe realizar es lo siguiente:

1. Abrir la terminal.
2. Posicionarnos en el directorio en el que se encuentra el archivo con extensión *.l con los comandos correspondientes.
3. Teclear el comando *flex *.l*; siendo * el nombre del programa con extensión *.l.
4. Teclear el comando *gcc lex.yy.c -ll* (en el caso de Mac) o *gcc lex.yy.c -lfl* (en el caso de Windows y Linux). El archivo lex.yy.c se generó mediante el comando del paso 3.
5. Teclear el comando *./a.out *.c*; siendo * el nombre del programa en C que se le dará como entrada al analizador para que pueda ejecutarse.
6. El paso anterior generará diferentes archivos de salida; uno que contiene los errores identificados, otro que contiene las tablas generadas (estáticas y dinámicas), y por último otro que posee los tokens generados.

Conclusiones

- **German López Rodrigo**

  Al finalizar la creación y el diseño de nuestro Analizador Léxico podemos afirmar que se cumplió el objetivo, el cual era reforzar y entender el funcionamiento de un Analizador Léxico así como las tareas que debe llevar acabo, durante el desarrollo del programa llegue a la conclusión que las expresiones regulares fueron vitales para el funcionamiento del analizador léxico, debido a que son las encargadas de validar que elementos debe reconocer el Analizador Léxico, dichas expresiones fueron creadas según las reglas que se establecieron en clase. También observe la complejidad e importancia de las estructuras de datos, ya que sin ellas no viera sido posible construir las Tablas Dinámicas que necesita el Analizador. Por ultimo pero no menos importante se puede concluir que lex es una herramienta que nos permitió la creación del analizador léxico de una manera más simple y eficaz.
  Aritmético

- **Lemuz Fuentes Omar Alejandro**

  Cumplir con el objetivo planteado no fue una tarea fácil, ya que para esto se requería la construcción total del analizador léxico, es decir, aplicar todo lo que se vio en clase con respecto a este en un lenguaje de programación.
  Uno de los problemas que se nos presentó al momento de realizar el programa fue la construcción de las expresiones regulares. En ocasiones tuvieron que probarse distintas alternativas hasta observar que la alternativa usada fuera la correcta; sin embargo, una vez probada la versión final de nuestro conjunto de expresiones nos dimos cuenta de que, en efecto, el funcionamiento de estas era correcto. Aceptaban las palabras que debían aceptar y descartaban las que no. Otra cuestión que por lo menos para mí resultó un pequeño problema fue el uso de lenguaje C en la mayor parte del programa, ya que, después de no haber programado por mucho tiempo en este lenguaje, en ocasiones lograba confundirme un poco.
  Por último, considero que es muy importante conocer cuál es el funcionamiento de este analizador, ya que, al ser uno de los componentes de un compilador, el hecho de

que funcione correctamente, o no, es crucial en el proceso de compilación; sin él los otros componentes no tendrían razón de ser y por supuesto, no funcionarían de manera correcta.