ANALIZADOR LÉXICO – SINTÁCTICO

Martínez Naredo Noé
07.nov.2017

Descripción del Problema:

En primera instancia, se pretende realizar un programa que lea un archivo y separe sus componentes léxicos en clases, así como descartar los comentarios. El analizador léxico tendrá como salida una tabla de tokens, una tabla de símbolos, una tabla de cadenas y una tabla de átomos. La siguiente tabla muestra las clases, con su descripción y su expresión regular.

0	Identificadores	[a-z]+
1	Palabras Reservadas	ENT|ESCRIBE|HAZ|LEE|MIENTRAS|REAL|SI|SINO
2	Operador de Asignación	\=
3	Operadores Relacionales	\=\=|<\=|<|>|>\=|!\=
4	Operadores Aritméticos	DIVIDE|MAS|MENOS|MULTIPLICA
5	Símbolos Especiales	[,;()\[\]]
6	Constantes numéricas enteras	[1-9]?[0-9]*
7	Constantes numéricas reales	[0-9]+[.][0-9]+
8	Constantes Cadenas	\".*\"

Para la parte del analizador sintáctico se utilizará la tabla de átomos que servirá para crear el árbol sintáctico, con el cual se determinará la validez del código con respecto al lenguaje definido en clase.

Propuesta de Solución:

Se comenzó por definir las estructuras de datos que contendrían las tablas. Se escogieron arreglos lineales (de una sola dimensión) de enteros .Al final la estructura de los tokens fue cambiada por una estructura de floats para facilitar el proceso de esta clase.

Los métodos de inserción consistieron en hacer una re-colocación de la estructura en la memoria reservando espacio para la estructura actual más lo que se fuera agregar. Por último, la estructura a agregar se unía al final de la primera.

Tuvieron que formularse métodos de búsqueda para las tablas de cadenas y de símbolos para validar cadenas ya existentes antes de su inserción. Para esto se usó aritmética de apuntadores así como variables enteras que también funcionaros como éstos (índices). Para distinguir las cadenas se uso de los delimitadores de cadena (carácter ‘\0’).

Una vez que se tuvo la primera parte, se crearon las funciones que servirían como manejador de casos. Se crearon también las funciones que mostraban las tablas y las imprimían en los archivos; para la fase del analizador léxico se muestra en pantalla el recorrido del análisis.

Una vez concluido el funcionamiento del programa en C se realizó el programa en Flex para el reconocimiento de los tokens, con las especificaciones antes descritas.

Por parte del Analizador Sintáctico, se diseñó una serie de funciones que corresponden a los terminales, con el propósito de recorrer sus producciones y mostrar los errores sintácticos.

Cómo correr el programa:

El programa tiene que compilarse, preferiblemente desde una distribución de Linux. Desde la terminal se ingresa al directorio donde se encuentra el archivo, o bien puede ejecutarse el programa flex proporcionándole como parámetro la ruta completa del archivo.

```
flex scanner.l
```

Con esto se obtendrá un archivo un archivo escrito en C de nombre *yy.lex.c* este archivo es un programa escrito en C. Este archivo tiene que ser compilado con el *gcc*, sin embargo hay que agregarle la función *–lfl* para que utilice las librerías de flex. Si no se cuenta con esta librería en las rutas de búsqueda de librerías de gcc habrá que explícitamente proporcionar la ruta, como en el siguiente ejemplo:

```
gcc -L '/Applications/XAMPP/xamppfiles/lib/' -lfl lex.yy.c
```

Una vez que se cuenta con el ejecutable, que por defaut recibe por nombre a.out, deberá de ejecutarse anteponiendo ./ (punto diagonal) y pasando como parámetro el archivo (o la ruta del archivo) del cual se van a extraer los token.

```
./a.out fuente.txt
```

Finalmente, el programa mostrará las tablas y crearán (en la carpeta donde se encuentre) los archivos símiles a las tablas.

Conclusiones:

La elaboración de este analizador léxico conllevó un análisis, en primer lugar, de las estructuras de datos. Se optó porque éstas fueran lineales (de una sola dimensión) debido a que así se podría tener un mejor control del manejo de la memoria de forma dinámica. Asegurando así que sólo se esté reservando la memoria necesaria y no más. Esto se traduce en mayor costo de las operaciones de búsqueda e inserción, pero se consideró que fuera más eficiente en este aspecto para que fuera capaz de procesar archivos de gran tamaño.

La planeación se enfoco primero a la construcción de estos arreglos y después en las funciones manejadores de las clases antes de definir las expresiones regulares y el resto del archivo flex. Después de la fase de pruebas, se encontraron algunas fallas, en la definición de los comentarios y en el manejador de casos de error. La presencia de comentarios hacia aparecer clases que no deseadas en la tabla de tokens. Se había omitido el manejador de errores ya que volvía al programa errático. En la segunda versión se estabilizaron estas incidencias.

Para la elaboración del análisis sintáctico, en primer lugar intenté diseñar la Tabla de Parser. Esto me tomó un tiempo considerable ya que poco a poco se volvió una tarea rebuscada. Creía que con esto se podía llegar a una implementación más puntual y eficiente. Tuve que retroceder lo avanzado para hacerlo con funciones (como se había visto en clase).

Al final se logró un analizador sintáctico que pudo corregir los errores del analizador léxico, entregado anteriormente.