Descripción del problema

Analizador Léxico

La primera fase de un compilador es el analizador léxico, el cual recibe como entrada un código fuente, para realizar un análisis e identificar aquellos elementos que pertenezcan al alfabeto definido, al igual que detectar los errores léxicos, la salida de este debe ser una lista de tokens, los cuales son una secuencia de componentes léxicos que siguen reglas léxicas expresadas mediante expresiones regulares.

En este caso las reglas léxicas se deben definir para el lenguaje de programación definido en clase. Dicho lenguaje tiene ocho distintas clases las cuales se enlistan en la siguiente tabla:

Clase	Descripción
0	Palabras reservadas
1	Identificadores (Solo letras, inician con mayúsculas y hasta 8 letras)
2	Símbolos especiales
3	Operadores de asignación
4	Operadores de relación
5	Operadores aritméticos
6	Constantes cadenas
7	Constantes numéricas
8	Constantes numéricas reales

Expresiones regulares usadas:

```
minuscula [a-z]
mayuscula [A-Z]
Letras ({minuscula}|{mayuscula}){0,7}
num [0-9]
rel DIF|IGL|MN|MNI|MY|MYI
punto \.
numEntero {num}+
numReal {num}+{punto}{num}+|{num}+{punto}{num}*|{num}*{punto}{num}+
exp (E|e)
signo (\+|-){0,1}
valExp {num}{1,2}

palRes Bul|Cadena|Cierto|Entero|Falso|Haz|Mientras|Para|Real|Si|Sino
iden {minuscula}{Letras}
simEsp (\(|\)|,|;|\[|\])
opAsig :=
opRel  {punto}{rel}{punto}
opArit (\+|-|\*|%|\/)
consCad  \".*\"
consNum {numEntero}|{numEntero}{exp}{signo}{valExp}
consNumReal {numReal}{exp}{signo}{valExp}|{numReal}
comentario \$\$.*
errorGeneral .
```

El analizador léxico también debe generar las tablas dinámicas y estáticas que se utilizarán en el programa, en este caso serán 3 tablas estáticas:

- Tabla de clases que debe tener el número de la clase, y su descripción
- Tabla de palabras reservadas que igualmente debe tener un número de identificador y la palabra reservada.
- Tabla de operadores relacionales, contiene dos campos: valor y operador relacional.

Tres tablas dinámicas:

- Tabla de símbolos donde se guardarán los identificadores, esta tabla contiene tres campos: Posición, identificador, y tipo este último vacío. Cada vez que se detecte un nuevo identificador se tiene que hacer una búsqueda.
- Tabla de cadenas, al llegar una cadena se debe guardar en esta tabla, la cual tiene dos campos: posición y cadena.
- Tabla de tokens, tiene dos espacios, clase a la que pertenece y el valor del token, este varía según la clase a la que pertenezca.

Análisis:

- Para el analizador léxico se deben definir las expresiones regulares para ello se transforman las especificaciones del lenguaje a expresiones regulares y posteriormente se llevan al lenguaje Lex.
- Una vez con todas las reglas se deben definir las tablas y métodos de búsqueda e inserción a emplearse.

Analizador Sintáctico

La segunda fase del compilador corresponde al analizador sintáctico, el cual recibe de entrada la salida del analizador léxico, en este caso se utilizó un analizador sintáctico descendente, para lo cual se necesitó una gramática libre de contexto (en este caso gramática LL). Esta parte del compilador se encarga de verificar que el código fuente siga la gramática previamente definida, así como también debe indicar los errores que posee el código fuente.

Para poder hacer el analizador sintáctico se necesita la definición de la gramática, es decir, como se deben estructurar las sentencias, una vez con la gramática, ésta se analiza para comprobar que pertenezca a la gramática que se necesita (gramática LL), posteriormente se lleva la gramática a un lenguaje de programación, con las respectivas consideraciones, lo más importante es adecuar el programa del analizador léxico para que se pueda hacer el análisis sintáctico.

Análisis al Analizador Léxico

Para el analizador léxico se deben definir las expresiones regulares para esto se deben transformar las especificaciones del lenguaje a expresiones regulares para posteriormente llevarlas a lenguaje Lex.

Una vez con todas las reglas se deben definir las tablas y métodos de búsqueda e inserción a emplearse. En este caso se usarán listas por lo que los métodos de búsqueda e inserción son para listas.

Actividades de cada integrante

Cuevas Salgado Carlos:

- Diseño e implementación de las expresiones regulares.
- Diseño de las pruebas de software.
- Pruebas de software en las expresiones regulares, en las tablas, métodos de búsqueda e inserción.
- Correcciones menores a partir de los resultados de las pruebas.

Chavez Galindo Lisset América:

- Diseño e implementación de las tablas dinámicas/estáticas.
- Método de búsqueda en las tablas estáticas basadas en arreglos.
- Métodos de búsqueda, inserción e impresión en las tablas estáticas, basadas en listas ligadas.
- Pruebas de software en las tablas, métodos de búsqueda e inserción.
- Correcciones menores a partir de los resultados de las pruebas.

Diseño e implementación:

Las tablas estáticas se definirán mediante un arreglo de apuntador, de la siguiente forma char *tablaEstatica = {valor1, valor2, … , valorn }
Para hacer la búsqueda en la tabla estática se implementará una función donde se mandará la tabla estática y el valor, se hará una comparación para encontrar el valor buscado con los valores de la tabla estática, y se retornará la posición indicada en la tabla estática.

Para las tablas dinámicas se usarán listas ligadas, en donde se mandará el valor a guardar. Para cada tabla se definirá su propia estructura, tanto para el nodo de cada una como para su lista enlazada, ya que los valores que utilizan son diferentes para cada una. La búsqueda en las listas se hace mandando el token y recorriendo toda la lista buscando el valor, si se encuentra en la lista no se guardará en caso contrario se almacenará al final de la lista.

Análisis al analizador Sintáctico

Para poder hacer el analizador sintáctico se necesita definir la gramática, analizar la gramática para comprobar que sea gramática LL y obtener los conjuntos de selección de cada producción, codificar la gramática usando lenguaje C, donde se debe adaptar el código previamente elaborado para que haga el análisis sintáctico, obtener la cadena de átomos que se analizara, elaborar la función para los errores.

El diseño de la gramática se hizo en conjunto con el resto del grupo, en donde se dividió al grupo en distintos equipos y a cada equipo se le asigno que definiera la gramática de una estructura de control. Posteriormente se hizo la integración de la gramática que hizo cada equipo para formar la gramática del lenguaje pu+. Una vez con la gramática cada equipo hizo el respectivo análisis de la gramática para determinar si era gramática LL.

A continuación se muestra la gramática obtenida, así como su conjunto de selección que nos servirá para poder programar la función correspondiente a cada terminal, necesario para poder realizar el analizador sintáctico

1: G→[Z]	27: K→s
2: Z→DZ	28: K→E
3: Z→ε	29: R→EQ
4: Z→Y	30: Q→OE
5: Y→SX	31: Q→ε
6: X→Y	32: O→!
7: X→ε	33: O→q
8: D→JaV	34: O→<
9: J→b	35: O→I
10: J→c	36: O→>
11: J→e	37: O→g
12: J→d	38: E→TE’
13: V→,aV;	39: E’→+TE’
14: V→;	40: E’→-TE’
15: S→A;	41: E’→ε
16: S→H	42: T→FT’
17: S→M	43: T’→*FT’
18: S→P	44: T’→/FT’
19: S→I	45: T’→%FT’
20: A→a=K	46: T’→ε
21: H→h[Y]m(R);	47: F→(E)
22: M→m(R)[Y]	48: F→a
23: P→p(A;R;A)[Y]	49: F→n
24: I→i(R)[Y]N	50: F→r
25: N→ε	51: K→t
26: N→o[Y]	52: K→f

a) Producciones anulables: 3, 7, 25, 31, 41, 46
No terminales anulables: Z, X, N, Q, E’, T’

b)

First(1) = { [}	First(27) = { s }
First(2) = { b c e d }	First(28) = { (a n r }
First(3) = { }	First(29) = { (a n r }
First(4) = { a h m p i }	

First(5) = { a h m p i } First(6) = { a h m p i } First(7) = { } First(8) = { b c e d } First(9) = { b } First(10) = { c } First(11) = { e } First(12) = { d } First(13) = { , } First(14) = { ; } First(15) = { a } First(16) = { h } First(17) = { m } First(18) = { p } First(19) = { i } First(20) = { a } First(21) = { h } First(22) = { m } First(23) = { p } First(24) = { i } First(25) = { } First(26) = { o }	First(30) = { ! q < I > g} First(31) = { } First(32) = { ! } First(33) = { q } First(34) = { < } First(35) = { I } First(36) = { > } First(37) = { g } First(38) = { (a n r } First(39) = { + } First(40) = { - } First(41) = { } First(42) = { (a n r } First(43) = { * } First(44) = { / } First(45) = { % } First(46) = { } First(47) = { (} First(48) = { a } First(49) = { n } First(50) = { r } First(51) = { t } First(52) = { f }

First(G) = { [} First(Z) = { b c e d a h m p i } First(Y) = { a h m p i } First(X) = { a h m p i } First(D) = { b c e d } First(J) = { b c e d } First(V) = { , ; } First(S) = { a h m p i } First(A) = { a } First(H) = { h } First(M) = { m } First(P) = { p }	First(I) = { i } First(N) = { o } First(K) = { s t f (a n r } First(R) = { (a n r } First(Q) = { ! q < I > g} First(O) = { ! q < I > g } First(E) = { (a n r } First(E') = { + - } First(T) = { (a n r } First(T') = { * / % } First(F) = { (a n r }

c)

Follow(G) = { }
Follow(Z) = {] }
Follow(Y) = Follow(Z) ∪ Follow(X) ∪ {] } = Follow(Z) ∪ Follow(Y) ∪ {] } = {] }
Follow(X) = Follow(Y) = {] }
Follow(D) = First(Z) U Follow(Z) = { b c e d a h m p i] }
Follow(J) = { a }
Follow(V) = Follow(D) = { b c e d a h m p i }
Follow(S) = First(X) U Follow(Y) = { a h m p i] }

Follow(A) = { ;) }
Follow(H) = Follow(S) = { a h m p i] }
Follow(M) = Follow(S) = { a h m p i] }
Follow(P) = Follow(S) = { a h m p i] }
Follow(I) = Follow(S) = { a h m p i] }
Follow(N) = Follow(I) = Follow(S) = { a h m p i] }
Follow(K) = { ;) }
Follow(R) = { ;) }
Follow(Q) = { ;) }
Follow(O) = First(E) = { (a n r }
Follow(E) = Follow(K) U First(Q) U Follow(R) U Follow(Q) U { (}
= { ;) ! q < I > g }
Follow(E') = Follow(E) = { ;) ! q < I > g }
Follow(T) = First(E') U Follow(E') = { + - ;) ! q < I > g }
Follow(T') = Follow(T) = { + - ;) ! q < I > g }
Follow(F) = First(T') U Follow(T') U Follow(T) = { * / % + - ;) ! q < I > g }

Conjuntos de selección

c.s.(1) = { [}	c.s.(27) = { s }
c.s.(2) = { b c e d }	c.s.(28) = { (a n r }
c.s.(3) = {] }	c.s.(29) = { (a n r }
c.s.(4) = { a h m p i }	c.s.(30) = { ! q < I > g}
c.s.(5) = { a h m p i }	c.s.(31) = { ;) }
c.s.(6) = { a h m p i }	c.s.(32) = { ! }
c.s.(7) = {] }	c.s.(33) = { q }
c.s.(8) = { b c e d }	c.s.(34) = { < }
c.s.(9) = { b }	c.s.(35) = { I }
c.s.(10) = { c }	c.s.(36) = { > }
c.s.(11) = { e }	c.s.(37) = { g }
c.s.(12) = { d }	c.s.(38) = { (a n r }
c.s.(13) = { , }	c.s.(39) = { + }
c.s.(14) = { ; }	c.s.(40) = { - }
c.s.(15) = { a }	c.s.(41) = { ;) ! q < I > g}
c.s.(16) = { h }	c.s.(42) = { (a n r }
c.s.(17) = { m }	c.s.(43) = { * }
c.s.(18) = { p }	c.s.(44) = { / }
c.s.(19) = { i }	c.s.(45) = { % }
c.s.(20) = { a }	c.s.(46) = { + - ;) ! q < I > g }
c.s.(21) = { h }	c.s.(47) = { (}
c.s.(22) = { m }	c.s.(48) = { a }
c.s.(23) = { p }	c.s.(49) = { n }
c.s.(24) = { i }	c.s.(50) = { r }
c.s.(25) = { a h m p i] }	c.s.(51) = { t }
c.s.(26) = { o }	c.s.(52) = { f }

Actividades de cada integrante:

Cuevas Salgado Carlos:

- Diseño de la gramática.
- Análisis de la gramática.
- Generación de la cadena de átomos.
- Codificación de la gramática mediante lenguaje C.
- Diseño e implementación de pruebas de software.
- Correcciones menores a partir de los resultados de las pruebas.

Chavez Galindo Lisset América:

- Diseño de la gramática.
- Análisis de la gramática.
- Codificación de la gramática mediante lenguaje C.
- Diseño e implementación de pruebas de pruebas de software.
- Correcciones menores a partir de los resultados de las pruebas.

Diseño e implementación:

Para la generación de la cadena de átomos se genero un apuntador tipo char el cual se utiliza para almacenar todos los átomos, para esto se utilizó el código escrito en lex, en la parte que se indica que hacer en cada expresión regular se le concatena al apuntador el nuevo átomo, se oficiaron las tablas para almacenar las palabras reservadas y los operadores relacionales para que tuvieran un nuevo campo donde se almacena la representación de la gramática de dicho campo.

Con la gramática comprobada que es de tipo LL se procedió a codificar cada producción, se generó una función por cada elemento no terminal, más la función principal donde se llama al elemento inicial de la gramática. Para facilitar el análisis sintáctico se generó la función Terminal (char terminal) donde se manda el elemento terminal a analizar y hace el procedimiento para validar dicho elemento.

Para los errores se generó la función error(), la cual no recibe ningún elemento, esta función imprime el átomo donde se detectó el error así como también se imprime el número de átomo donde se encontró el error.

Funcionamiento

Funcionamiento del programa, se debe compilar primero el archivo.l, después el archivo con extensión .yy.c y finalmente el ejecutable como se muestra en la imagen adicionalmente se debe enviar por línea de comando el archivo a analizar.

La salida serán dos archivos en uno se mostrarán los errores encontrados en el análisis y en el segundo se encontrará de manera general el token generado y su tipo, no se enviarán tablas a ningún archivo. En pantalla se observará las diferentes tablas solicitadas en el análisis léxico así como también se vera la cadena de átomos.

Conclusiones:

Cuevas Salgado Carlos

Al hacer el analizador léxico tuve mejor entendimiento del lenguaje Lex, además de como poder usarlo con lenguaje C ya que tenia ciertas dudas. La herramienta lex fue de gran ayuda para este programa ya que hizo los tokens y también para poder clasificar ciertos errores léxicos y mostrarlos al usuario. En la parte del llenado de las tablas hubo un problema con el apuntador yytext el cual enviaba información adicional además del token esto por un error en el concepto de apuntador que se resolvía con aun apuntador auxiliar dinámico.

Galindo Chávez Lisset América

Gracias a lo elaborado en este programa logré entender como es que trabaja un analizador léxico, desde que es lo que recibe como entrada hasta que retorna como salida. Además con el uso de Flex fue más sencillo desarrollar este programa, gracias a las herramientas que posee. Por otro lado, debido a que me toco la parte del desarrollo de tablas junto con su inserción, búsqueda e impresión, me di cuenta que para poder ingresar un texto obtenido a partir de yytext, no se puede hacer directamente ya que copia todo el texto y no únicamente la palabra, para solucionar esto se tuvo que usar una variable auxiliar para luego agregarse al nodo de la tabla correspondiente.

Conclusiones de analizador sintáctico

Cuevas Salgado Carlos

Se aplicaron los conocimientos vistos en clase además de los conocimientos de materias previas (principalmente lenguajes formales y autómatas), el poder hacer un analizador sintáctico reafirma estos conceptos, los cuales llegan a ser muy abstractos, ya al aplicarlos el llevarlos a un lenguaje de programación se logra una mejor comprensión de todo, además que es una muy buena practica ya que es un método de procesamiento de texto, la metodología aplicada fue sencilla, hacer la cadena de átomos y analizar dichos átomos mediante la gramática previamente definida.

Dentro de las principales complicaciones del analizador sintáctico se encuentra el definir la función de errores y aplicar las pruebas, referente a los errores fue complicado lograr un control de las líneas así como también hacer las pruebas puesto pese a que definimos el lenguaje no fue tan sencillo el adaptarse a este.

Galindo Chávez Lisset América

Gracias a la elaboración de este programa, me fue posible observar como trabaja el analizador sintáctico descendente, en este caso, con una gramática LL(1), para el cual obtuvimos su conjunto de selección y la función de cada no terminal. Lo cual

nos generó uno que otro error, no por la complejidad sino más bien por la cantidad de funciones que tuvimos que realizar y que además tuvimos que adaptar al analizador léxico que habíamos realizado anteriormente.

Por otro lado, la parte que más trabajo nos costó fue al momento de decidir como íbamos a tratar los errores, optamos por llevar un contador para indicar el número de linea y mandar el átomo donde se genere el error, cabe mencionar que si el error es la falta de un ; el error lo marca en la linea siguiente.

Otra parte que también nos fue difícil de realizar fueron las pruebas, ya que nos dimos cuenta que en el caso del condicional Si después de este necesariamente debe seguirlo un Sino, una asignación o un ciclo porque al pasar el conjunto de selección de N cuando produce ε, olvidamos colocar ‘]’, lo cual me dejo observar la importancia que tienen los conjuntos de selección y como un error tan sencillo puede generar un cambio total en este nuevo lenguaje de programación.