

Universidad Nacional Autónoma de México

Facultad de Ingeniería

División de Ingeniería Eléctrica

Compiladores

Compilador PU+

Integrantes:

Cuevas Salgado Carlos

Galindo Chávez Lisset América

Grupo: 2

Profesora: Sandoval Montaña Laura Ing.

Semestre 2019-1

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.

Analizador Semántico

La tercera fase de un compilador es el analizador semántico, en esta parte se hace el correspondiente análisis para verificar que posea un sentido. A partir de la gramática ya definida en el análisis sintáctico se hacen las correspondientes modificaciones para que se pueda realizar el análisis semántico añadiendo los correspondientes atributos y símbolos de acción para poder realizar la traducción solicitada. Cabe destacar que al agregar símbolos de acción y atributos no alteran los conjuntos de selección.

Este análisis se hará a la par del análisis sintáctico, por lo que se modificaran las funciones ya creadas con anterioridad. Se completará el llenado de la tabla símbolos donde tipo donde a su campo *tipo* se indicará que tipo de valor es el identificador (Bul, Entero, Real, Cadena). En caso de que una variable ya este declarada y se intente declararla nuevamente con otro tipo se indicara que existe un error. ADICIONAL si una variable no se encuentra declara y se utiliza en alguna parte del programa se indicará un error puesto no se declaró.

Análisis

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

24: $l \rightarrow i(R)[Y]N$	50: $F \rightarrow r$
25: $N \rightarrow \epsilon$	51: $K \rightarrow t$
26: $N \rightarrow o[Y]$	52: $K \rightarrow f$

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

First(G) = { [}	First(l) = { i }
First(Z) = { b c e d a h m p i }	First(N) = { o }

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(K) = { s t f (a n r }

First(R) = { (a n r }

First(Q) = { ! q < l > g }

First(O) = { ! q < l > 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) U Follow(X) U {] } = Follow(Z) U Follow(Y) U {] } = {] }

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 < l > g }

Follow(E') = Follow(E) = { ;) ! q < l > g }

Follow(T) = First(E') U Follow(E') = { + - ;) ! q < l > g }

Follow(T') = Follow(T) = { + - ;) ! q < l > g }

Follow(F) = First(T') U Follow(T') U Follow(T) = { * / % + - ;) ! q < l > 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 < l > 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) = { l }
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 < l > 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 < l > 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 software.

Correcciones menores a partir de los resultados de las pruebas.

Diseño e implementación:

Para la generación de la cadena de átomos se generó 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.

Análisis al Analizador Semántico

En esta parte del análisis, es primordial colocar de manera adecuada los símbolos de acción y los atributos ya que son los que ayudaran a la traducción del programa. En todos los terminales de la gramática se agregaron símbolos de acción ya que se mandarán al archivo de traducción, para los operadores relacionales los símbolos de acción mandan la traducción solicitada, en la producción de declaración se colocó un símbolo de acción para no perder el tipo de la variable. En algunos; [] () se mandaron tabulaciones y/o salto de línea.

A continuación se muestra la gramática con símbolos de acción y atributos:

1: $G \rightarrow [\{ \} Z] \{ \}$	27: $K \rightarrow s \{ s \}$
--------------------------------------	-------------------------------

2: $Z \rightarrow DZ$	28: $K \rightarrow E$
3: $Z \rightarrow \varepsilon$	29: $R \rightarrow EQ$
4: $Z \rightarrow Y$	30: $Q \rightarrow OE$
5: $Y \rightarrow SX$	31: $Q \rightarrow \varepsilon$
6: $X \rightarrow Y$	32: $O \rightarrow \{!\}$
7: $X \rightarrow \varepsilon$	33: $O \rightarrow q\{q\}$
8: $D \rightarrow J a_p \{AT\}_{t1p1} V_{t2}$	34: $O \rightarrow \{<\}$
9: $J \rightarrow b_b \{b\}$	35: $O \rightarrow l\{l\}$
10: $J \rightarrow c_c \{c\}$	36: $O \rightarrow \{>\}$
11: $J \rightarrow e_e \{e\}$	37: $O \rightarrow g\{g\}$
12: $J \rightarrow d_d \{d\}$	38: $E \rightarrow TE'$
13: $V_t \rightarrow \{, \} a_p \{AT\}_{t1p1} V_{t2}; \{;\}$	39: $E' \rightarrow \{+\} TE'$
14: $V \rightarrow \{;\}$	40: $E' \rightarrow \{-\} TE'$
15: $S \rightarrow A; \{;\}$	41: $E' \rightarrow \varepsilon$
16: $S \rightarrow H$	42: $T \rightarrow FT'$
17: $S \rightarrow M$	43: $T' \rightarrow \{* \} FT'$
18: $S \rightarrow P$	44: $T' \rightarrow \{/ \} FT'$
19: $S \rightarrow I$	45: $T' \rightarrow \% \{ \} FT'$
20: $A \rightarrow a \{a\} \{=\} K$	46: $T' \rightarrow \varepsilon$
21: $H \rightarrow h \{h\} \{ \{ \} Y \{ \} \} m \{m\} \{ \{ \} R \{ \} \};$	47: $F \rightarrow \{ \{ \} E \{ \} \}$
22: $M \rightarrow m \{m\} \{ \{ \} R \{ \} \} \{ \{ \} Y \{ \} \}$	48: $F \rightarrow a \{a\}$
23: $P \rightarrow p \{p\} \{ \{ \} A; \{;\} R; \{;\} A \{ \} \} \{ \{ \} Y \{ \} \}$	49: $F \rightarrow n \{n\}$
24: $I \rightarrow i \{i\} \{ \{ \} R \{ \} \} \{ \{ \} Y \{ \} \} N$	50: $F \rightarrow r \{r\}$
25: $N \rightarrow \varepsilon$	51: $K \rightarrow t \{t\}$
26: $N \rightarrow o \{o\} \{ \{ \} Y \{ \} \}$	52: $K \rightarrow f \{f\}$

Actividades de cada integrante:

Cuevas Salgado Carlos:

Adecuaciones a la gramática.

Codificación de la gramática mediante lenguaje C.

Diseño e implementación de pruebas de software.

Correcciones menores a partir de los resultados de las pruebas.

Galindo Chavez Lisset América:

Adecuaciones a la gramática.

Codificación de la gramática mediante lenguaje C.

Diseño e implementación de pruebas de software.
Correcciones menores a partir de los resultados de las pruebas.

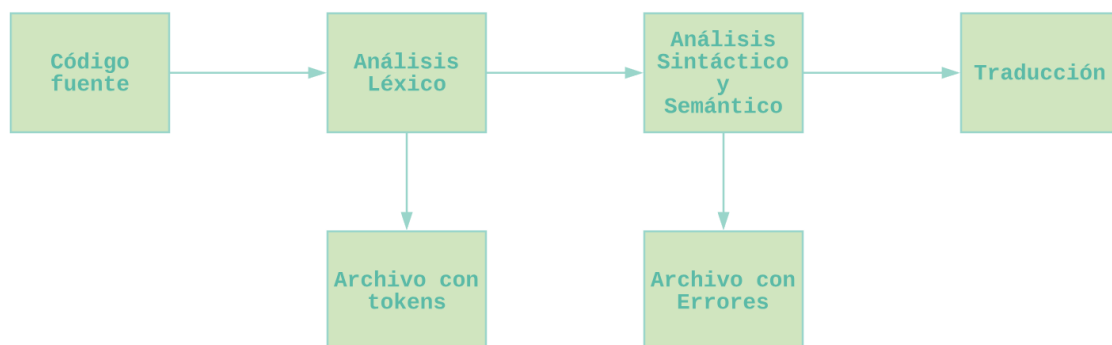
Diseño e implementación:

Se necesitaron crear dos funciones para poder hacer el correspondiente análisis, la primera para poder obtener el cambio de tipo en la tabla de símbolo y poder modificarlo, la segunda es para buscar cadenas y poder mandarlas a la traducción.

Las funciones que se tuvieron que modificar fueron las que conformaban el análisis sintáctico, todo esto hacer el análisis simultáneamente. Básicamente las instrucciones agregadas eran para moverse en la lista de tokens, e imprimir en el archivo de salida la correcta traducción.

Para las correspondientes funcionalidades para indicar si un identificador ya estaba asignado y lo que se realizó ADICIONALMENTE para señalar si una variable no estaba declarada. Primero se revisó en que producción podía surgir dicha situación, posteriormente con una función creada en esta etapa fue de gran ayuda para poder resolver este problema.

Arquitectura



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.

```
Compiladores]$ flex Programal.l  
Compiladores]$ gcc lex.yy.c -lfl  
Compiladores]$ ./a.out Prueba.c
```

La salida serán tres archivos en uno se mostrarán los errores encontrados en el análisis, en el segundo se encontrará de manera general los tokens generados no, finalmente en el tercero se encontrará la traducción solicitada del programa. En ningún archivo se enviarán tablas generadas. En pantalla se observará las diferentes tablas solicitadas, así como indicaciones de cuando inicia cada análisis, la cadena de átomos y los errores que puedan surgir.

Conclusiones

Conclusiones de analizador léxico

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 línea 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 línea 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.

Conclusiones de analizador semántico

Cuevas Salgado Carlos

Lo primero que tuvimos que resolver fue como colocar correctamente los símbolos de acción y atributos, por lo que se le dedico cierto tiempo para que la gramática quedara de manera adecuada. Sin esta parte no se hubiera podido hacer de manera correcta el analizador. Para poder hacer que la traducción saliera con una indentación adecuada fue necesario pensar de que manera poder solucionar esto, lo cual llevo cierto tiempo.

Respecto a la implementación del código, si se tuvo que pensar bien en que forma agregar y modificar las funciones puesto se estaban ocupando las mismas funciones para el análisis sintáctico, y crear las funciones indispensables en esta parte del análisis las cuales son las de buscar símbolo y buscar cadena, ambas funciones fueron muy útiles.

Finalmente, la parte mas complicada de un compilador es su diseño ya que a partir del diseño es como se podrá implementar y ocasionará que su complejidad de resolución del problema se eleve o disminuya. Así mismo la parte que requería mas tiempo para su implementación fue el analizador léxico porque se generaron las tablas, y la parte que mas tiempo se requiero pensar en su diseño fue la gramática en la parte del sintáctico, la dificultad del semántico radica en ir siguiendo la forma que se necesitaran los símbolos de acción y los atributos.

Galindo Chávez Lisset América

Para poder realizar de forma correcta esta parte semántica primero tuvimos que agregarle a la gramática los respectivos símbolos de acción junto con sus atributos, lo cual fue fundamental para poder traducir correctamente la traducción, para esto fue necesario repasar los conceptos vistos en clase, así como comprender claramente cada concepto.

Una vez que se obtuvo correctamente la gramática se procedió por agregarlo al programa, haciendo más sencillo poder incluir tanto los símbolos de acción correspondientes como los atributos, a las funciones del analizador sintáctico recursivo para poder aplicar acertadamente la traducción. Además se tuvo que crear otra función que regresara el símbolo según la posición dada en la tabla de símbolos, en un principio únicamente se mandaba el nombre del símbolo, pues es lo que necesita el traductor, fue necesario mandar la estructura del símbolo como

tal, ya que necesitábamos saber si ya se había especificado su tipo, tanto para saber si se vuelve a declarar así como para saber que se está utilizando se haberse declarado. Cabe mencionar que la traducción la mandamos a un archivo para mayor comodidad.

Por último, creo que la parte más difícil en cuanto a la programación fue darle un buen formato a la traducción, porque se debía lograr una buena indentación, por ello, para solucionar este problema se agregó un contador que indicara el número de tabuladores que se necesitaban y una función que recibiera este contador para después imprimir estos tabuladores en el archivo que contiene el programa traducido.