Análisis léxico-sintáctico

Objetivos

a) Construir un analizador Sintáctico Descendente Recursivo

Descripción

Construir, en un mismo programa, los analizadores Léxico y Sintáctico Descendente Recursivo que revisen programas escritos en el lenguaje definido por la gramática dada en anexo A.

- La entrada es un archivo con el programa fuente a analizar que deberá estar escrito en el lenguaje definido por la gramática. Este archivo de entrada se indicará desde la línea de comandos.
- El programa realizará tanto el análisis léxico como el sintáctico. Para esto, en su Analizador Léxico del proyecto 1 se deben incluir todas las funciones del Analizador Sintáctico Recursivo en la zona de funciones auxiliares. Además, en la función main( ) deberán llamar, después de yylex() a la función correspondiente al símbolo inicial de la gramática.

Definición de expresiones regulares para los componentes léxicos

%%

Acciones

%%

```
main(…)
{
   …
   yylex( );
   Programa( );
   …
}
```

Funciones del Analizador Sintáctico.

- El analizador léxico deberá generar además de los tokens, la cadena de átomos que será la entrada del analizador sintáctico. Los átomos se pueden ir generando a la par que los tokens, pero irlos almacenando en una sola cadena.

- Los átomos estarán definidos por cada componente léxico y corresponden a los elementos terminales de la gramática. La tabla de clases de componentes léxicos con sus correspondientes átomos es:

Clase	Descripción	átomo
0	Palabras reservadas (ver tabla)	(ver tabla)
1	Operadores aritméticos + - * / $	Mismo operador
2	Operadores de asignación (ver tabla)	(ver tabla)
3	Símbolos especiales ( ) { } [ ] & , :	Mismo símbolo
4	Operadores relacionales (ver tabla)	(ver tabla)
5	Identificadores. Inicien con letra (mayúscula o minúscula), le sigan letras o dígitos (hasta cinco) y terminen con guion bajo.	a
6	Constantes numéricas enteras (base 10, sin sufijos) signados o no signados, máximo 6 dígitos, mínimo 1. Para no confundir el signo con un operador aritmético se dejará o no un espacio en los siguientes casos: -56+ +46 56- 41 72+a_	z
7	Constantes numéricas reales. No signados. Casos: 2.67 .85 31. Al definir la expresión regular, no debe incluir la cadena . como constante real.	r
8	Constantes cadenas. Iniciar y terminar con comillas dobles, cualquier carácter excepto las comillas dobles. Tamaño entre 2 y 40 (considerando las comillas).	s
9	Constante carácter. Cualquier carácter encerrado entre comillas sencillas (apóstrofos).	c

- El valor en los tokens y los átomos se indican en las siguientes tablas.

Valor	Palabra reservada	átomo
0	cadena	h
1	caracter	g
2	else	e
3	entero	n
4	for	f
5	if	i
6	real	d
7	return	b
8	void	v
9	while	w

Valor	Op. relacional	significado	átomo
0	^^	mayor que	>
1	^”	menor que	<
2	==	igual que	?
3	^^=	mayor o igual	y
4	^”=	menor o igual	l
5	<>	diferente	¿

Valor	Op. asignación	significado	
0	~	igual	=
1	+~	más igual	m
2	-~	menos igual	k
3	*~	por igual	p
4	/~	entre igual	t
5	$~	módulo igual	u

- El analizador sintáctico deberá mostrar todos los errores sintácticos que encuentre, indicando qué se esperaba.
- **Como resultados, el analizador léxico-sintáctico deberá mostrar el contenido de la tabla de símbolos, las tablas de literales, los tokens y la cadena de átomos. Finalmente deberá indicar si está sintácticamente correcto el programa fuente.**
- Los errores que vaya encontrando el analizador léxico, los podrá ir mostrando en pantalla o escribirlos en un archivo, así como él o los errores sintácticos. Es conveniente que **cuando encuentre un error sintáctico se indique en qué átomo de la cadena se encontró (con ubicación).**
- El programa deberá estar comentado, con una descripción breve de lo que hace, el nombre de quienes elaboraron el programa y fecha de elaboración.

Entregar:

Un documento con la siguiente estructura:

- Descripción del problema (no del programa).
- **El conjunto de selección de cada producción;** si resultan conjuntos de selección no disjuntos para producciones de un mismo no-terminal, hacer los ajustes pertinentes para que resulte una gramática LL(1) y explicarlo claramente en el documento.
- Indicaciones de cómo correr el programa.
- Conclusiones de cada uno de los integrantes del equipo.

Enviar el documento, el programa fuente definitivo y archivo de prueba en la plataforma en un archivo zip

Fecha de entrega: 30 de octubre de 2024.

Anexo A

La gramática que define la sintaxis del lenguaje se muestra a continuación.

Sintaxis de las diferentes estructuras del lenguaje

1)Sentencia declarativa:

Ejemplos:

caracter x_~'h ':
real a1_, a2_, b3x_~3.62:

Gramática:

D → <Tipo>L:
<Tipo>→ g
<Tipo>→n
<Tipo>→d
<Tipo>→h
L → a<Valor> I'
I'→,a<Valor> I'
I'→ ε
<Valor> → = V
<Valor>→ ε
V→c
V→s
V→z
V→r

2)Sentencias de asignación:

Ejemplos: a1_+~ 2:
b3x_~5.3*(a1_+a2_):

Gramática:

A -> a A':
A' -> = E
A' -> m E
A' -> k E
A' -> p E
A' -> t E
A' -> u E

3)Sentencia if

Ejemplo:

```
if [<expRel>] (
    <sentencias>
)
else (
    <sentencias>
)
```

Gramática:

I → i [R] F'
F' → (S)B
B → e(S)
B→ ε

4)Sentencia while

Ejemplo:

```
while [<expRel>](
    <sentencias>
)
```

Gramática:

W -> w[R](S)

5)Sentencia for

Ejemplo:

```
for [E] (
    <sentencias>
)
```

Gramática:

<For> → f [E](S)

6)Sentencia return:

Ejemplo:

return [E]:

return:

Gramática:

<Return> → bZ

Z → [E]:

Z →:

7) Expresión aritmética

E ⟶ T E’

E’ ⟶ + T E’

E’ ⟶ - T E’

E’ ⟶ ξ

T ⟶ F T’

T’ ⟶ *F T’

T’ ⟶ /FT’

T’ ⟶ $FT’

T’ ⟶ ξ

F ⟶ (E)

F ⟶ a

F ⟶ z

F ⟶ r

F ⟶ [a]

8) Expresion relacional

Ejemplos:

2 ^” c_

a1_ +b_*3 ^^= r2cc_

Gramática:

R → ER’

R’ → > E

R’ → < E

R’ → ? E

R’ → y E

R’ → | E

R’ → ¿ E

9) Sentencias

S ⟶ S’<otraS>

S’ ⟶ A

S’ ⟶ I

S’ ⟶ W

S’ ⟶ <For>

S’ ⟶ <Return>

<otraS> ⟶ S’<otraS>

<otraS> ⟶ ξ

10) Funciones

Ejemplos:

void fun1_ (

<ListaD>

<sentencias>

)

Gramática:

<Func> ⟶ <TipoFun>a(<ListaD>S)

<TipoFun> ⟶ <Tipo>

<TipoFun> ⟶ v

11) Llamada a una función

[fun1_]

Gramática:

S’ ⟶ [a]:

Estructura del Programa:

<Declaraciones globales>

<Serie de funciones>

Gramática:

<Programa> ⟶ <ListaD>[<SerieF>]

<ListaD> ⟶ D<ListaD>

<ListaD> ⟶ ξ

<SerieF> ⟶ <Func><otraF>

<otraF> ⟶<Func><otraF>

<otraF> ⟶ ξ

Compiladores Semestre 2022-1

Gramática del lenguaje

1:	<Programa> →<ListaD>[<SerieF>]
2:	<ListaD> → D<ListaD>
3:	<ListaD> → ξ
4:	<SerieF> → <Func><otraF>
5:	<otraF> →<Func><otraF>
6:	<otraF> → ξ
7:	D → <Tipo>L:
8:	<Tipo>→ g
9:	<Tipo>→n
10:	<Tipo>→d
11:	<Tipo>→h
12:	L → a<Valor> I’
13:	I’→,a<Valor> I’
14:	I’→ ξ
15:	<Valor> → = V
16:	<Valor>→ ξ
17:	V→c
18:	V→s
19:	V→z
20:	V→r
21:	A -> a A’:
22:	A’ -> = E
23:	A’ -> m E
24:	A’ -> k E
25:	A’ -> p E
26:	A’ -> t E
27:	A’ -> u E
28:	A -> a A’:
29:	I → i [R] F’
30:	F’ → (S)B
31:	B → e(S)
32:	B→ ε
33:	I → i [R] F’
34:	W -> w[R](S)
35:	<For> → f [E](S)

36:	<Return> → bZ
37:	Z → [E]:
38:	Z →:
39:	E → TE’
40:	E’ → +TE’
41:	E’ → -TE’
42:	E’ → ξ
43:	T → FT’
44:	T’ → *FT’
45:	T’ → /FT’
46:	T’ → $FT’
47:	T’ → ξ
48:	F → (E)
49:	F → a
50:	F → z
51:	F → r
52:	F → [a]
53:	R → ER’
54:	R’ → >E
55:	R’ → <E
56:	R’ → ?E
57:	R’ → yE
58:	R’ → |E
59:	R’ → ¿E
60:	S → S’<otraS>
61:	S’ → A
62:	S’ → I
63:	S’ → W
64:	S’ → <For>
65:	S’ → <Return>
66:	S’ → [a]:
67:	<otraS> → S’<otraS>
68:	<otraS> → ξ
69:	<Func> → <TipoFun>a(<listaD>S)
70:	<TipoFun> → <Tipo>
71:	<TipoFun> → v