Compiladores Semestre 2023-1

Segundo Programa
Análisis léxico-sintáctico

Objetivo

Construir, en un mismo programa, los analizadores Léxico y Sintáctico Descendente Recursivo que revisen programas escritos en el lenguaje definido por la gramática del Anexo A de este documento.

Descripción

- La entrada es un archivo con el programa fuente a analizar que deberá estar escrito en el lenguaje definido por la gramática del Anexo A de este documento. Este archivo de entrada se indicará desde la línea de comandos.
- El programa realizará tanto el análisis léxico como el sintáctico (en el Anexo B encontrarás cómo incluirlo en el analizador léxico). 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 están definidos en este documento 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	Identificadores. Iniciar con $ y le sigue al menos una letra minúscula o mayúscula. Ejemplos: $ejemplo, $Varible, $OtraVariable, $XYZ	i
2	Constantes numéricas enteras. En base 10 (secuencia de dígitos del 0-9 sin 0's a la izquierda, excepto para el número 0), en base 8 (inicien con O u o y le sigan dígitos del 0 al 7).	n
3	Constantes numéricas reales. Siempre deben llevar parte decimal y es opcional la parte entera. Ejemplos: 73.0, .0, 10.2 No aceptados: **.** , 12 , 4**.**	r
4	Constantes cadenas. Encerrado entre comillas (**"**) cualquier secuencia de más de un carácter que no contenga **"** ni **'**. Para cadenas de un solo carácter, encerrarlo entre apóstrofos ('). La cadena de unas comillas debe ser encerrada entre apóstrofos: **'"'** . La cadena de un apóstrofo debe ser encerrada por comillas: **"'" .** No se aceptan cadenas vacías. Ejemplos NO válidos: "ejemplo no "valido" , "" , """ , "hola 'mundo"	s
5	Símbolos especiales **[ ] ( ) { } , : ;**	mismo símbolo
6	Operadores aritméticos **+ - * / % \ ^**	mismo símbolo
7	Operadores relacionales (ver tabla).	(ver tabla)
8	Operador de asignación =	=

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

Valor	Palabra reservada	Equivale en C	átomo
0	alternative	case	a
1	big	long	b
2	evaluate	if	f
3	instead	else	t
4	large	double	g
5	loop	while	w
6	make	do	m
7	number	int	#
8	other	default	o
9	real	float	x
10	repeat	for	j
11	select	switch	h
12	small	short	p
13	step	continue	c
14	stop	break	q
15	symbol	char	y
16	throw	return	z

Valor	Op. relacional	átomo
0	<	<
1	>	>
2	<=	l
3	>=	u
4	==	e
5	!=	d

- 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 (puede ser el objetivo indicado en este documento), el nombre de quienes elaboraron el programa y fecha de elaboración.

Entregar:

Un documento con la siguiente estructura:

- Descripción del problema (no del programa).
- Si se entrega en equipo, indicar qué actividad hizo cada uno.

- **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.

Nota: se podrá elaborar individualmente o en equipo de 2

Enviar el documento y sólo el programa fuente definitivo en la tarea de la plataforma Chamilo correspondiente, en un archivo zip o rar y en su caso, sólo un miembro del equipo. También sería conveniente enviar un archivo fuente de prueba (dentro del zip o rar).

Fecha de entrega: 6 de diciembre de 2022.

Anexo A

De acuerdo con el ejercicio realizado por todo el grupo, la gramática que define la sintaxis de nuestro lenguaje se muestra a continuación.

Sintaxis de las diferentes estructuras del lenguaje

1) Sentencia declarativa:

Ejemplos:

```
big  $Grande=2, $num;
real $numReal;
symbol  $cad=”cadena#1”, $cadVacia;
```

Gramática:

D → <Tipo>K;	Q → =NC
<Tipo>→ b	Q → ,K
<Tipo>→ g	N → n
<Tipo>→ #	N → r
<Tipo>→ y	N → s
<Tipo>→ x	C → ξ
K → iQ	C → ,K
Q → ξ	

2) Sentencias de asignación:

Ejemplos:

```
$num=$Grande+15*(o12-93);
$cadVacia= “cadena#2”;
$numReal=27.9;
```

Gramática:

A → i=A’;
A’ → s
A’ → E

3) Expresión aritmética

Ejemplos:

34.7*25 (12.6+$num)/[$calcula(63)]

E → T E’	T’ → %FT’
E’ → + T E’	T’ → ^FT’
E’ → - T E’	T’ → ξ
E’ → ξ	F → (E)
T → F T’	F → i
T’ → *F T’	F → n
T’ → /FT’	F → r
T’ → \FT’	F →<Llama>

4) Expresión relacional

Ejemplos:

```
2 <= $num
$cad==”cadena#2”
40.8>$numReal
```

Gramática:

R → iR’V	V → i
R → nR’V’	V → n
R → rR’V’’	V → r
R → sR’V’’’	V → s
R’ → >	V’ → n
R’ → <	V’ → i
R’ → l	V’’ → r
R’ → e	V’’ → i
R’ → d	V’’’ → s
R’ → u	V’’’ → i

5) Proposiciones

P → A	P → <Llama>
P → I	P → <Devuelve>
P → H	P → c;
P → W	
P → J	

6) Lista de 0 más proposiciones:

<listaP> → ξ
<listaP> → P<listaP>

7) Sentencia Loop

Ejemplo:

```
loop ($num>6) make
 {
     <proposiciones>
 }
```

Gramática:

W → w(R)m{<listaP>}

8) Sentencia Evaluate

Ejemplos:

```
evaluate (<expRel>)
    <proposiciones>
instead
    <proposiciones>
:

evaluate (<expRel>)
    <proposiciones>
:
```

Gramática:

I → f(R)<listaP>I':
I' → t<listaP>
I' → ξ

9) Sentencia Repeat

Ejemplo:

```
repeat($n=6; $n<10; $n=$n+1)
{
    <proposiciones>
}
```

Gramática

J → j(YXZ{<listaP>}
Y → i=E;
Y → ;
X → R;
X → ;
Z → i=E)
Z →)

10) Sentencia Select

Ejemplo:

```
select($num){
  case 1: <proposiciones>
          stop
  case 2: <proposiciones>
  other: <proposiciones>
}
```

Gramática:

H→ h(i){C'O'}
C' → an:<listaP>UC'
C' → ξ
O' → o:<listaP>
O' → ξ
U → q
U → ξ

11) Sentencia Throw

Ejemplos:

```
throw($valor);
throw("cadena1");
throw();
```

Gramática:

<Devuelve> → z(<valor>);
<valor> → V
<valor> → ξ

12) Llamada a una función

Ejemplo:

```
[$fun($x,8)]
```

Gramática:

<Llama> → [i(<arg>)]
<arg> → ξ
<arg> → V<otroArg>
<otroArg> → ,V<otroArg>
<otroArg> → ξ

13) Funciones

Ejemplo:

```
number  $fun (real $x, number $n){
    <sentencias>
}
```

Gramática:

<Func> → <Tipo>i(<Param>){<Cuerpo >}
<Param> → <Tipo>i<otroParam>
<Param> → ξ
<otroParam>→,<Tipo>i<otroParam>
<otroParam> → ξ
<Cuerpo> → <Decl><listaP>
<Decl> → ξ
<Decl> → D<Decl>

Estructura del Programa:

<Serie de funciones>

Gramática:

<Program> $\rightarrow$<Func><otraFunc>

<otraFunc> $\rightarrow$<Func><otraFunc>

<otraFunc> $\rightarrow$ ξ

Gramática del lenguaje

1:	<Program> →<Func><otraFunc>
2:	<otraFunc> →<Func><otraFunc>
3:	<otraFunc> → ξ
4:	<Func> → <Tipo>i(<Param>){<Cuerpo >}
5:	<Param> → <Tipo>i<otroParam>
6:	<Param> → ξ
7:	<otroParam>→,<Tipo>i<otroParam>
8:	<otroParam> → ξ
9:	<Cuerpo> → <Decl><listaP>
10:	<Decl> → ξ
11:	<Decl> → D<Decl>
12:	D → <Tipo>K;
13:	<Tipo>→ b
14:	<Tipo>→ g
15:	<Tipo>→ #
16:	<Tipo>→ y
17:	<Tipo>→ x
18:	K → iQ
19:	Q → ξ
20:	Q → =NC
21:	Q → ,K
22:	N → n
23:	N → r
24:	N → s
25:	C → ξ
26:	C → ,K
27:	A → i=A’;
28:	A’ → s
29:	A’ → E
30:	E → T E’
31:	E’ → + T E’
32:	E’ → - T E’
33:	E’ → ξ
34:	T → F T’
35:	T’ → *F T’
36:	T’ → /FT’
37:	T’ → \FT’
38:	T’ → %FT’
39:	T’ → ^FT’
40:	T’ → ξ
41:	F → (E)

42:	F → i
43:	F → n
44:	F → r
45:	F → <Llama>
46:	R → iR’V
47:	R → nR’V’
48:	R → rR’V’’
49:	R → sR’V’’’
50:	R’ → >
51:	R’ → <
52:	R’ → l
53:	R’ → e
54:	R’ → d
55:	R’ → u
56:	V → i
57:	V → n
58:	V → r
59:	V → s
60:	V’ → n
61:	V’ → i
62:	V’’ → r
63:	V’’ → i
64:	V’’’ → s
65:	V’’’ → i
66:	P → A
67:	P → I
68:	P → H
69:	P → W
70:	P → J
71:	P → <Llama>
72:	P → <Devuelve>
73:	P → c;
74:	<listaP> → ξ
75:	<listaP> → P<listaP>
76:	W → w(R)m{<listaP>}
77:	I → f(R)<listaP>I’:
78:	I’ → t<listaP>
79:	I’ → ξ
80:	J → j(YXZ{<listaP>}
81:	Y → i=E;
82:	Y → ;

83:	X → R;
84:	X → ;
85:	Z → i=E)
86:	Z →)
87:	H→ h(i){C'O'}
88:	C' → an:<listaP>UC'
89:	C' → ξ
90:	O' → o:<listaP>
91:	O' → ξ
92:	U → q

93:	U → ξ
94:	<Devuelve> → z(<valor>);
95:	<valor> → V
96:	<valor> → ξ
97:	<Llama> → [i(<arg>)]
98:	<arg> → ξ
99:	<arg> → V<otroArg>
100:	<otroArg> → ,V<otroArg>
101:	<otroArg> → ξ

Anexo B

El requisito de la entrega de este programa es que en un solo archivo se encuentren los Analizadores Léxico y Sintáctico. Para esto, en su Analizador Léxico deberán 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.

Estructura del programa a entregar (tendrá la extensión .l)

```
Definición de expresiones regulares para los componentes léxicos
%%
Acciones
%%
main(…){
    …
    yylex( );
    Program( );
    …
}
Funciones del Analizador Sintáctico.
```

Nota: Recuerden que para que funcione bien su Analizador Sintáctico, debe funcionar bien el Analizador Léxico. Espero que ya hayan atendido las observaciones que les hice al revisarles su Analizador Léxico.