Compiladores Semestre 2021-1

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	Constantes numéricas reales Ej: 2.17, 0.39, 2.7E-3, 3.5E5, 18.0e+5	r
1	Palabras reservadas (ver tabla)	Ver tabla
2	Símbolos especiales { } , : ( ) [ ]	{ } , : ( ) [ ]
3	Identificadores. Letra mayúscula o minúscula y puede estar seguida de letras mayúsculas o minúsculas, dígitos o guiones bajos.	a
4	Operador de asignación =	=
5	Operadores relacionales (ver tabla)	Ver tabla
6	Constantes numéricas enteras (base 10, sin sufijos y hasta 14 dígitos)	e
7	Cadenas. Cualquier número de caracteres encerrados entre comillas (“”), incluye cadena vacía.	s
8	Operadores aritméticos + - * / ** %	+ - * / # %

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

Valor	Palabra reservada	átomo
0	bool	b
1	break	q
2	case	k
3	char	c
4	continue	t
5	default	d
6	do	h
7	else	l
8	float	f

Valor	Palabra reservada	átomo
9	for	p
10	if	i
11	int	n
12	return	u
13	string	g
14	switch	x
15	while	w
16	void	o

Valor	Op. Relacional	átomo
0	!=	!
1	==	?
2	>	>
3	<	<
4	>=	y
5	<=	m

- 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).
- **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: 3 de diciembre de 2020.

Anexo A

De acuerdo al ejercicio realizado por todo el grupo, la gramática que define la sintaxis de nuestro lenguaje se muestra a continuación (hago notar que revisando las palabras reservadas, faltaban definir las sentencias *for, continue, return* y llamada a una función; así como el uso de *void*, y definir las estructuras de las funciones y del programa fuente):

Sintaxis de las diferentes estructuras del lenguaje

Declaración de variables:

<Tipo> <lista_identif>:

Ejemplos:

```
int a[3],b,d:
bool c:
char x,y,z:
float r1,h[5]:
string cadena[4], cad:
```

Gramática:

D → V L:
L → aGC
C →,L
C →ξ
V → b
V → c
V → f
V → n
V → g
G → [e]
G → ξ

Sentencia de Asignación:

<identificador> = <expr.aritm>:
<identificador> =<cte cadena>:
<identificador> = +<lista_cadenas>:
<identificador> = [<llamada_fun>]:

Ejemplos

```
c=1:
b=2*c/3**5:
a[3]=b:
d=[funcionX()]:
cadena[1] = “cadena1”:
cadena[2] = “cadena2”:
cadena[0] = cadena[1]:
cadena[3]=+ “cadena3”,cadena[2]:
cad=+ cadena[1], “uno”, “otracadena”:
```

Observa que en la lista de cadenas para la concatenación puede haber identificadores y constantes cadenas. *Elaborar la gramática.*

Sentencia Switch:

```
switch (<identificador>)
{
        case <const_entera>[<sentencias>]
        break:
        . . .
        default[<sentencias>]
}
```

Ejemplo:

```
switch (menu)
{
        case 1 [ cad = “Se eligio opcion 1”]
        break:
        case 2 [ cad = “Se eligio opcion 2”]
        break:
        default [ cad = “Elige una opcion”]
}
```

Gramática:

X → x(a){Od[B]}
O →ke[B]q:O
O → ξ
B →SB
B → ξ

Sentencia if:

```
if (<expr. relacional>)
{
        <sentencias>
 }
 else
{
        <sentencias>
 }
```

Gramática:

I → i(R){B}J :
J → l{B}
J → ξ

Sentencia while

```
 while (<expr. relacional>)
{
     <sentencias>
}
```

Gramática:

W→w(R){B}

Sentencia do while

```
do {
    <sentencias>
} while (<exp. relacional>):
```

Gramática:

H → h{B}w(R):

Sentencia for

```
for [<num_veces>]
{
     <sentencias>
 }
```

Gramática:

N → p[e]{B}

Expresión relacional:

< expr.aritm > <op rel> < expr.aritm >

Gramática:

R →EKE
K → !
K → ?
K → >
K → <
K → y
K → m

Expresión aritmética:

Ejemplos:

a	a*(r+a)**e
e	a+[a()]%e
r	e-a[e]/[a()]
[a()]	e*r-(a[e]+r)

Gramática:

E → T E'
E' → + T E'
E' → - T E'
E' → ξ
T→ F T'
T' → *F T'
T' → /F T'
T' → %F T'
T' → #F T'
T' → ξ
F → (E)
F → aG
F → e
F → r
F → [a()] llamada a una función

Sentencias
Son: asignación, switch, if, while, do while, for, return, continue y llamada a una función

S → A
S → X
S → I
S → W
S → H
S → N
S → uU:
S → t:
S → [a()]:
U → ξ
U →(F)

Funciones
Estructura de las funciones:

```
 [ <Tipo_función> <nombre_fun> ()
    {
       <Declaraciones>
       <sentencias>
     }
  ]
```

Ejemplo:

```
  [ void funcion1()
    {
      int a,b:
      a=12:
      b=a%3:
     }
   ]
```

Gramática:

Y →[V'a(){D'B}]
V' → V
V' → o
D' → DD'
D' → ξ

Estructura del Programa Fuente:

Declaraciones
Grupo de funciones

Gramática: (P es el símbolo inicial de la gramática)
P → D'YY'
Y' → YY'
Y' → ξ

Nota: Como no definimos el componente léxico 'constante carácter' ni los valores booleanos FALSE y TRUE, Los valores de las variables tipo "char" se darán como el ASCII del carácter y las variables booleanas tendrán valor de 0 o 1 (esto no se revisa en el análisis sintáctico pero sí en el semántico que después haremos).
Ejemplos:

char c1,c2:
bool bandera:
c1=65:
bandera=0:

Gramática del lenguaje

1:	P → D'YY'	33:	X → x(a){Od[B]}
2:	Y' → YY'	34:	O →ke[B]q:O
3:	Y' → ξ	35	O → ξ
4:	Y →[V'a(){D'B}]	36	B →SB
5:	V' → V	37	B → ξ
6:	V' → o	38	I → i(R){B}J :
7:	D' → DD'	39	J → l{B}
8:	D' → ξ	40	J → ξ
9:	D → V L:	41	N → p[e]{B}
10:	L → aGC	42	R →EKE
11:	C →,L	43	K → !
12:	C →ξ	44	K → ?
13:	V → b	45	K → >
14:	V → c	46	K → <
15:	V → f	47	K → y
16:	V → n	48:	K → m
17:	V → g	49	E → T E'
18:	G → [e]	50	E' → + T E'
19:	G → ξ	51	E' → - T E'
20:	S → A	52	E' → ξ
21:	S → X	53	T → F T'
22:	S → I	54	T' → *F T'
23:	S → W	55	T' → /F T'
24:	S → H	56	T' → %F T'
25:	S → N	57	T' → #F T'
26:	S → uU:	58	T' → ξ
27:	S → t:	59	F → (E)
28:	S → [a()]:	60	F → aG
29:	U → ξ	61	F → e
30:	U →(F)	62	F → r
31:	W→w(R){B}	63	F → [a()]
32	H → h{B}w(R):	64	A→

Para que esté completa la gramática, define e incluye las producciones que definen la sentencia de asignación (A)

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 P( ).

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

```
Definición de expresiones regulares para los componentes léxicos
%%
Acciones
%%
main(…){
    …
    yylex( );
    P( );
    …
}
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.