MEMORIA PRÁCTICA PROCESADORES DE LENGUAJE

PECL

11 DE DICIEMBRE DE 2016

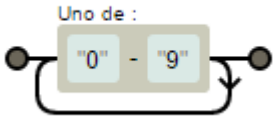
RICARDO SÁNCHEZ MALO
0072163C

ANALIZADOR LÉXICO

Para esta parte de la practica he creado un programa .lex en el que se reconocen tres tipos diferentes de autómatas, los identificadores, los números y los errores de reconocimiento. Aparte de esto, se introducen las palabras reservadas que se usaran.

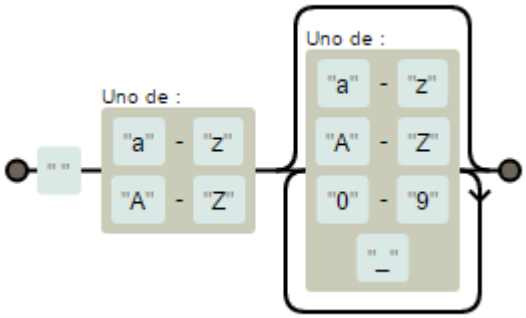
Definimos los numero:

Integer =[0-9]+



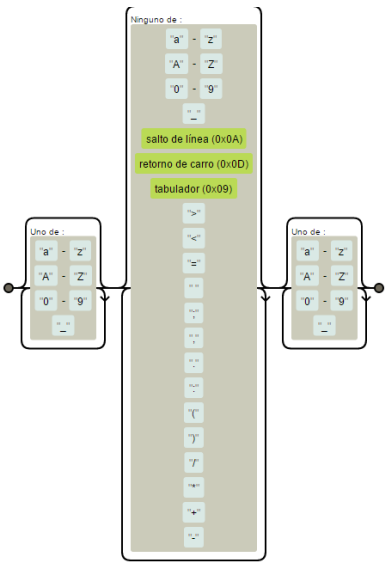
Definimos los identificadores:

Identifier = [a-zA-Z][a-zA-Z0-9_]*



Definimos los errores de asignación:

ErrorAsign1 = [a-zA-Z0-9_]*[^a-zA-Z0-9_\n\r\t><= ;,.:()/*+-]*[a-zA-Z0-9_]*



EspLex 1	Requisito cumplido	Correcto 1,2,3.
EspLex 2	Requisito cumplido	Correcto 1,2,3.
EspLex 3	Requisito cumplido	Correcto 1,2,3.
EspLex 4	Requisito cumplido	Correcto 1,2,3.
EspLex 5	Requisito cumplido	Correcto 1,2,3.
EspLex 6	Requisito cumplido	Correcto 1,2,3.
EspLex 7	Requisito cumplido	Correcto 1,2,3.
EspLex 8	Requisito cumplido	Correcto 1,2,3.
EspLex 9	Requisito cumplido	Correcto 1,2,3.
ReqLex 1	Requisito cumplido	Correcto 1,2,3. Incorrecto 4
ReqLex 2	Requisito cumplido	Hecho arriba
ReqLex 3	Requisito cumplido	Explicación abajo
ReqLex 4	Requisito cumplido	Todos

JUSTIFICACIÓN DEL USO O NO DE LOS ESTADOS LÉXICOS

No he utilizado estados léxicos debido a que me ha parecido más sencillo pasar los tokens completos a la interfaz con la que interactúan, y creando un token de errores en el que entra todo lo recogido todo lo que no era válido. De esta forma con tres autómatas finitos deterministas recojo todas las opciones que no son palabras reservadas.

ANALIZADOR SINTÁCTICO

He construido la siguiente gramática:

```
start with inicio;
inicio          ::=PROGRAM nombreProg IS variables programa ;
nombreProg      ::=IDE;
variables       ::=variables variable;
variable        ::=VAR enum DOS_PUNTOS var PUNTO_COMA
                 | VAR:e enum PUNTO_COMA   | error;
enum            ::=IDE:ID | enum COMA IDE:
                 | error ;
var             ::=INTEGER
                 | BOOLEAN;
programa        ::=BEGIN sentencias END  ;
sentencias      ::=expresion mas | ;
mas             ::=PUNTO_COMA expresion mas | ;
expresion       ::=  operación
                  | asignacion
                  | if
                  | while
                  | expresion2 ;
asignacion      ::= IDE: ASIGNACION  tipo
                  | error tipo;
tipo            ::= IDE
                  | numero
                  | tipoBool ;
tipoBool        ::= TRUE{:RESULT="TRUE"; :}
                  | FALSE{:RESULT="FALSE"; :}
                  | NOT PARENT_IZQ logica PARENT_DER ;
comparacion     ::= IDE comp IDE:  | IDE comp numero
                  | IDE IGUAL tipoBool
                  | error;
comp            ::= MENOR_IGUAL
                  | MENOR
                  | IGUAL
                  | MAYOR
                  | MAYOR_IGUAL
                  | DISTINTO ;
logica          ::= IDE
                  | comparacion;
exLogicas       ::= logica
                  | NOT PARENT_IZQ exLogicas PARENT_DER |
exLogicas tLogica otra ;
otra            ::= exLogicas;
tLogica         ::= AND
                  | OR;
```

```
parentExLogica  ::= exLogicas
                  | PARENT_IZQ exLogicas PARENT_DER;
if              ::= IF parentExLogica THEN sentencias END IF
                  | IF parentExLogica THEN else ELSE sentencias END IF;



else            ::= sentencias;
while           ::= WHILE parentExLogica DO sentencias END WHILE ;
operacion       ::= IDE ASIGNACION opcion  | error opcion   ;
opcion          ::= operaciones
                  | PARENT_IZQ operaciones PARENT_DER;
operaciones     ::= operador operando operadorParent opcion2
                  | operando operador operando operadorParent opcion2
                 | error;
opcion2         ::= opcion |;
operando        ::= IDE
                  | numero;
operadorParent  ::= operador
                  |;
operador        ::= SUMA
                  | RESTA
                  | DIVISION
                  | MULTIPLICACION;
conSigno        ::= SUMA NUM | RESTA NUM;
numero          ::= conSigno  | NUM;
skip            ::= SKIP;
read            ::= READ IDE;
write           ::= WRITE IDE
                  | WRITE opcion;
expresion2      ::= skip
                  | read
                  | write
                  | error;
```

EspSimp 1	Requisito cumplido	Correcto 1,2,3. Incorrecto 4.
EspSimp 2	Requisito cumplido	Correcto 1,2,3. Incorrecto 5, 6.
EspSimp 3	Requisito cumplido	Correcto 1,2,3. Incorrecto 3.
EspSimp 4	Requisito cumplido	Incorrecto 1,2,3.
EspSimp 5	Requisito cumplido	Correcto 1,2,3. Incorrecto 4.
EspSimp 6	Requisito cumplido	Correcto 1,2,3. Incorrecto 3.
EspSimp 7	Requisito cumplido	Correcto 1,2,3.
EspSimp 8	Requisito cumplido	Correcto 1,2,3.
ReqSimt 1	Requisito cumplido	Correcto 1,2,3.
ReqSimt 2	Requisito cumplido	Correcto 1,2,3. Explicado abajo.
ReqSimt 3	Requisito cumplido	Correcto 1,2,3. Incorrecto 4
ReqSimt 4	Requisito cumplido	Correcto 3. Incorrecto 1,2 ,3 4,5
ReqSimt 5	Requisito cumplido	Incorrecto 1,2,3,4,5.
ReqSimt 6	Requisito incompleto	Correcto 1,2,3. Explicado abajo

SOLUCIÓN A LA AMBIGÜEDAD

Al desarrollar la gramática he tenido problemas de ambigüedades, tanto en el caso del if-else como en el caso de operacion que ha sido resuelto utilizando el mismo mecanismo, creando un no terminal auxiliar, que en el que se asocia una de las expresiones ambiguas por lo que de esta manera se resuelve esta situación.

SOLUCIÓN AL PROBLEMA DE LOS SIGNOS

Inicialmente la parte léxica distinguía entre números con signo, o sin signo. Esto hacia que el analizador sintáctico se encontrara un signo, y siempre lo cogía como suma o resta. La solución elegida, ha sido la de distinguir si es un numero con signo o sin signo en la parte sintáctica, de tal manera que permitiera coger un signo cuando esperaba una suma o resta y permitiera también coger un signo en el no terminal numero.

ARBOL SINTÁCTICO

Para realizar el árbol sintáctico he creado 5 clases, una árbol, que recibe un nodo como raíz en el constructor, y e implementa los métodos para imprimirlo. Una clase abstracta nodo en la que se implementa los métodos de añadir hijo, y situar y devolver los valores. Y tres clases herederas de nodo que implementan los métodos que vayan a usarse. Estas clases representan la raíz, las hojas y los niveles intermedios del árbol. Y lo más complicado ha sido introducir los nodos, ya que el árbol fue inicialmente pensado para introducir primero los padres y luego los hijos, y el analizador sintáctico ejecuta antes el código de los terminales a lo de los no terminales. Por lo que he tenido que ir añadiendo los nodos por debajo, utilizando nodos auxiliares. Debido a estos problemas, no he sido capaz de añadir las sentencias de los if y while.

COMPROBACIÓN DE .PROG

Para la comprobación de que el programa sea introducido con la extensión correcta se crea una función en el parser code. Esta función analiza el parámetro que entra como argumento y separa sus últimos cinco caracteres. Una vez separados estos últimos caracteres se comprueba si son iguales o no a .prog.

ANALIZADOR SEMÁNTICO

TABLA DE SIMBOLOS

Para realizar el analizador semántico es necesario la creación de una serie de métodos en los que se apoya en la tabla de símbolos. Para la tabla de símbolos hemos creado un objeto Símbolo, en el que se almacena el tipo del símbolo (integer o boolean) y si esta inicializado o no.

La tabla de símbolos, consiste en un hashtable que asocia un string que guarda el nombre del símbolo con el objeto símbolo.

FUNCIONES AUXILIARES

Para el analizador semántico es necesario crear una serie de funciones auxiliares en las que se acceda a la tabla de símbolos. Se puede asociar dos clases. Las que añaden datos a la tabla de símbolos y las que la consultas. Todas estas funciones están añadidas en el action code.

Los que añaden datos, simplemente comprueban que no esté introducido el valor ya en la tabla. Por lo que se añade en primer lugar el nombre del programa y luego el de las variables.

En los que acceden a la base de datos, hay varios tipos de funciones. Unas que comprueban si los identificadores son un numero o un booleano y si están inicializados y otras que comprueban dos identificadores y comprueban que sean del mismo tipo y que estén inicializados. El último tipo es el de asignación en las que además de comprobar la concordancia de los tipos de identificadores o si es del tipo adecuado el identificador que se asigna, se va a marcar como inicializado. En las funciones de asignación si el tipo es undefined se asigna a que sea el tipo de datos que le entra.

EspSemp 1	Requisito cumplido	Correcto 1,2,3.
EspSemp 2	Requisito cumplido	Correcto 1,2,3. Incorrecto 4.
EspSemp 3	Requisito cumplido	Correcto 1,2,3.
EspSemp 4	Requisito cumplido	Correcto 1,2,3. Incorrecto 1.
EspSemp 5	Requisito cumplido	Correcto 1,2,3. Incorrecto 1, 3, 5.
EspSemp 6	Requisito cumplido	Correcto 1,2,3. Incorrecto 3, 4.
EspSemp 7	Requisito cumplido	Correcto 1. Incorrecto 2.
EspSemp 8	Requisito cumplido	Correcto 2,3.
EspSemp 9	Requisito cumplido	Incorrecto 2, 4.
EspSemp 10	Requisito cumplido	Correcto 1,2,3.
EspSemp 11	Requisito incompleto	Correcto 1, 2 3.
EspSemp 12	Requisito incumplido	Si comprueba los undefined. Incorrecto 3.
EspSemp 13	Requisito cumplido	Incorrecto 1, 2, 3.

ANALIZADOR SINTÁCTICO

Programa Correcto 1:

```
program prog is
var min, h, a :boolean;
var p, b  :integer;
var mas, mu;
var sd:boolean;
var in,dop :integer;
begin
a:=true;
b:=1;
p:=(5+6/b*9);
h:=false;
b:= 765;
skip;
while (a=true) do
h:=false
end while;
read p
end
```

Salida:

```
....ANALISIS FINALIZADO .....
Numero de errores sintacticos: 0
Numero de errores semanticos: 0
TABLA DE SIMBOLOS:
NOMBRE       TIPO         INICIALIZADO
p            integer      true
dop          integer      false
min          boolean      false
prog         pseudo       false
h            boolean      true
sd           boolean      false
in           integer      false
b            integer      true
a            boolean      true
mas          undefined    false
mu           undefined    false

ARBOL SINTACTICO:
program([Decl [boolean min (), h (), a (), ]]
[Decl [integer p (), b (), ]]
[Decl [undefined mas (), mu (), ]]
[Decl [boolean sd (), ]]
[Decl [integer in (), dop (), ]]
[Asign [a BOOL (TRUE), ]]
[Asign [b NUM (1), ]]
[Operation [p [PARENT op (5+6/6), ]]]
[Asign [h BOOL (FALSE), ]]
[Asign [b NUM (765), ]]
[skip  (), ]
[while [IF Log (a igual TRUE), ]]
[read IDE (p), ]
)
```

ANALIZADOR LEXICO

```
[program, ide (prog), is, var, ide (min),
coma, ide (h), coma, ide (a), dos_puntos,
boolean, punto_coma, var, ide (p), coma,
ide (b), dos_puntos, integer, punto_coma, var,
ide (mas), coma, ide (mu), punto_coma, var,
ide (sd), dos_puntos, boolean, punto_coma, var,
ide (in), coma, ide (dop), dos_puntos, integer,
punto_coma, begin, ide (a), asign, true,
punto_coma, ide (b), asign, num (1), punto_coma,
ide (p), asign, paren_izq, num (5), mas,
num (6), div, ide (b), mul, num (9),
paren_der, punto_coma, ide (h), asign, false,
punto_coma, ide (b), asign, num (765), punto_coma,
skip, punto_coma, while, paren_izq, ide (a),
comp, true, paren_der, do, ide (h),
asign, false, end, while, punto_coma,
read, ide (p), end]

Numero de errorres lexicos: 0
```

Programa Correcto 2:

```
program prog is
var min, h, a :boolean;
var b  :integer;
begin
a:=true;
b:=1;
h:=false;
b:= 765;
if (a=true) then
h:=false;
skip
end if;
write b
end
```

Salida:

```
....ANALISIS FINALIZADO .....
Numero de errores sintacticos: 0
Numero de errores semanticos: 0
TABLA DE SIMBOLOS:
NOMBRE       TIPO         INICIALIZADO
b            integer            true
a            boolean            true
min          boolean            false
h            boolean            true
prog         pseudo       false
```

```
ARBOL SINTACTICO:
program([Decl [boolean min (), h (), a (), ]]
[Decl [integer b (), ]]
[Asign [a BOOL (TRUE), ]]
[Asign [b NUM (1), ]]
[Asign [h BOOL (FALSE), ]]
[Asign [b NUM (765), ]]
[Asign [h BOOL (FALSE), ]]
[skip  (), ]
[if [IF Log (a igual TRUE), ]]
[write IDE (b), ]
)
ANALIZADOR LEXICO

[program, ide (prog), is, var, ide (min),
coma, ide (h), coma, ide (a), dos_puntos,
boolean, punto_coma, var, ide (b), dos_puntos,
integer, punto_coma, begin, ide (a), asign,
true, punto_coma, ide (b), asign, num (1),
punto_coma, ide (h), asign, false, punto_coma,
ide (b), asign, num (765), punto_coma, if,
paren_izq, ide (a), comp, true, paren_der,
then, ide (h), asign, false, punto_coma,
skip, end, if, punto_coma, write,
ide (b), end]

Numero de errorres lexicos: 0
```

Programa Correcto 3:

```
program prog is
var min, h, a :boolean;
var b  :integer;
begin
a:=true;
b:=1;
h:=false;
b:= 765;
if not (a) then
h:=false;
skip
end if;
write a
end
```

Salida:

```
....ANALISIS FINALIZADO .....
Numero de errores sintacticos: 0
Numero de errores semanticos: 0
TABLA DE SIMBOLOS:
NOMBRE       TIPO         INICIALIZADO
b            integer            true
a            boolean            true
min          boolean            false
h            boolean            true
prog         pseudo       false
```

```
ARBOL SINTACTICO:
program([Decl [boolean min (), h (), a (), ]]
[Decl [integer b (), ]]
[Asign [a BOOL (TRUE), ]]
[Asign [b NUM (1), ]]
[Asign [h BOOL (FALSE), ]]
[Asign [b NUM (765), ]]
[Asign [h BOOL (FALSE), ]]
[skip  (), ]
[if [IF Log (a), ]]
[write IDE (a), ]
)
ANALIZADOR LEXICO

[program, ide (prog), is, var, ide (min),
coma, ide (h), coma, ide (a), dos_puntos,
boolean, punto_coma, var, ide (b), dos_puntos,
integer, punto_coma, begin, ide (a), asign,
true, punto_coma, ide (b), asign, num (1),
punto_coma, ide (h), asign, false, punto_coma,
ide (b), asign, num (765), punto_coma, if,
not, paren_izq, ide (a), paren_der, then,
ide (h), asign, false, punto_coma, skip,
end, if, punto_coma, write, ide (a),
end]

Numero de errorres lexicos: 0
```

Programa InCorrecto 1:

```
program prog is
var min, h, a :boolean;
var b  :integer;
begin
a:=true;
b:=1;
h:=false;
b:= 765;
if not (a>3) then
h:=3;
skip
end if;
write
end
```

Salida:

```
Error semantico en la linea 9. La variable a no es compatible con integer
Error semantico en la linea 10. La variable h no es compatible con integer
Error de sintaxis linea 14 en o cerca de end

....ANALISIS FINALIZADO .....
Numero de errores sintacticos: 1
Numero de errores semanticos: 2
```

```
ANALIZADOR LEXICO

[program, ide (prog), is, var, ide (min),
coma, ide (h), coma, ide (a), dos_puntos,
boolean, punto_coma, var, ide (b), dos_puntos,
integer, punto_coma, begin, ide (a), asign,
true, punto_coma, ide (b), asign, num (1),
punto_coma, ide (h), asign, false, punto_coma,
ide (b), asign, num (765), punto_coma, if,
not, paren_izq, ide (a), comp, num (3),
paren_der, then, ide (h), asign, num (3),
punto_coma, skip, end, if, punto_coma,
write, end]

Numero de errorres lexicos: 0
```

Programa InCorrecto 2:

```
program prog is
var min, h, a :boolean;
var b  :integer;
begin
a:=true;
prog:=1;
h:=false;
b:= 765;
read h
end
```

Salida:

```
Error semantico en la linea 6. La variable prog es el nombre del programa, no
puede ser usado como variable
Error semantico en la linea 9. La variable h debe ser un entero

....ANALISIS FINALIZADO .....
Numero de errores sintacticos: 0
Numero de errores semanticos: 1

ANALIZADOR LEXICO

[program, ide (prog), is, var, ide (min),
coma, ide (h), coma, ide (a), dos_puntos,
boolean, punto_coma, var, ide (b), dos_puntos,
integer, punto_coma, begin, ide (a), asign,
true, punto_coma, ide (prog), asign, num (1),
punto_coma, ide (h), asign, false, punto_coma,
ide (b), asign, num (765), punto_coma, read,
ide (h), end]

Numero de errorres lexicos: 0
```

Programa InCorrecto 3:

```
program prog is
var min, h, a :boolean;
var b  :integer;
begin
a:=true;
b:=1;
h:=false;
b:= 765;
if (a=false and b) then
h:=min
end if;
read a
end
```

Salida:

```
Error semantico en la linea 9. La variable b debe ser  boolean
Error semantico en la linea 10. La variable min no esta inicializada
Error semantico en la linea 12. La variable a debe ser un entero

....ANALISIS FINALIZADO .....
Numero de errores sintacticos: 0
Numero de errores semanticos: 1

ANALIZADOR LEXICO

[program, ide (prog), is, var, ide (min),
coma, ide (h), coma, ide (a), dos_puntos,
boolean, punto_coma, var, ide (b), dos_puntos,
integer, punto_coma, begin, ide (a), asign,
true, punto_coma, ide (b), asign, num (1),
punto_coma, ide (h), asign, false, punto_coma,
ide (b), asign, num (765), punto_coma, if,
paren_izq, ide (a), comp, false, and,
ide (b), paren_der, then, ide (h), asign,
ide (min), end, if, punto_coma, read,
ide (a), end]

Numero de errorres lexicos: 0
```

Programa InCorrecto 4:

```
program prog is
var min, h, a :bool$ean;
var b, prog :integer;
begin
a:=true
b:=1;
h:=7;
b:= 765;
if (a=false and b) then
h:=min
end if;
read a
end
```

Salida:

```
Asignador incorrecto en linea: 2 columna:16 Carácter: bool$ean

Error de sintaxis linea 2 en o cerca de ;
Error semantico en linea 3. El caracter prog ya esta definido
Error de sintaxis linea 6 en o cerca de b
Error semantico en la linea 9. La variable a no es compatible con boolean
Error semantico en la linea 9. La variable b debe ser  boolean
Error semantico en la linea 10. La variable h no es compatible con min

....ANALISIS FINALIZADO .....
Numero de errores sintacticos: 2
Numero de errores semanticos: 3

ANALIZADOR LEXICO

[program, ide (prog), is, var, ide (min),
coma, ide (h), coma, ide (a), dos_puntos,
punto_coma, var, ide (b), coma, ide (prog),
dos_puntos, integer, punto_coma, begin, ide (a),
asign, true, ide (b), asign, num (1),
punto_coma, ide (h), asign, num (7), punto_coma,
ide (b), asign, num (765), punto_coma, if,
paren_izq, ide (a), comp, false, and,
ide (b), paren_der, then, ide (h), asign,
ide (min), end, if, punto_coma, read,
ide (a), end]

Numero de errorres lexicos: 1
```

Programa InCorrecto 5:

```
program prog2 is
var x,y,b: integer;
var b,c: boolean;
begin
read x; read c; write x+a;
b := x < c;
if x = y
then c := x <= y
else y := x > y
end if;
while c > b do x := x + 1 end while
end
```

Salida:

```
Error semantico en linea 3. El caracter b ya esta definido
Error semantico en la linea 5. La variable x no esta inicializada
Error semantico en la linea 5. La variable c no esta inicializada
Error semantico en la linea 5. La variable x no esta inicializada
Error semantico en la linea 5. La variable a no esta defninida
Error de sintaxis linea 6 en o cerca de <
Error semantico en la linea 7. La variable x no esta inicializada
Error semantico en la linea 7. La variable y no esta inicializada
Error de sintaxis linea 8 en o cerca de <=
Error semantico en la linea 8. La variable c debe ser un entero
```

```
Error de sintaxis linea 9 en o cerca de >
Error semantico en la linea 11. La variable c no es compatible con b
Error semantico en la linea 11. La variable x no esta inicializada

....ANALISIS FINALIZADO .....
Numero de errores sintacticos: 3
Numero de errores semanticos: 8

ANALIZADOR LEXICO

[program, ide (prog2), is, var, ide (x),
coma, ide (y), coma, ide (b), dos_puntos,
integer, punto_coma, var, ide (b), coma,
ide (c), dos_puntos, boolean, punto_coma, begin,
read, ide (x), punto_coma, read, ide (c),
punto_coma, write, ide (x), mas, ide (a),
punto_coma, ide (b), asign, ide (x), comp,
ide (c), punto_coma, if, ide (x), comp,
ide (y), then, ide (c), asign, ide (x),
comp, ide (y), else, ide (y), asign,
ide (x), comp, ide (y), end, if,
punto_coma, while, ide (c), comp, ide (b),
do, ide (x), asign, ide (x), mas,
num (1), end, while, end]

Numero de errorres lexicos: 0
```

Programa InCorrecto 6:

```
program frombinary is
var sum, n : integer;
var n : boolean;
begin
bol := true;
sum := 10; read n;
while ( n < 2 ) do
sum := 2*3+n; read n
end while;
write sum;skip
end
```

Salida:

```
Error semantico en linea 3. El caracter n ya esta definido
La variable bol no esta defninida
Error semantico en la linea 6. La variable n no esta inicializada
Error semantico en la linea 7. La variable n no esta inicializada
Error semantico en la linea 8. La variable n no esta inicializada
Error de sintaxis linea 8 en o cerca de ;
Error semantico en la linea 8. La variable n no esta inicializada

....ANALISIS FINALIZADO .....
Numero de errores sintacticos: 1
Numero de errores semanticos: 5
```

ANALIZADOR LEXICO

```
[program, ide (frombinary), is, var, ide (sum),
coma, ide (n), dos_puntos, integer, punto_coma,
var, ide (n), dos_puntos, boolean, punto_coma,
begin, ide (bol), asign, true, punto_coma,
ide (sum), asign, num (10), punto_coma, read,
ide (n), punto_coma, while, paren_izq, ide (n),
comp, num (2), paren_der, do, ide (sum),
asign, num (2), mul, num (3), mas,
ide (n), punto_coma, read, ide (n), end,
while, punto_coma, write, ide (sum), punto_coma,
skip, end]

Numero de errorres lexicos: 0
```

Programa incorrecto 7: programa.pog

```
program inc is
begin
end
```

Salida:

```
ERROR. EL ARCHIVO DEBE DE SER .prog
```