Practica 2.- Analizador Léxico de Simplon

TALF 2016/17

6 de marzo de 2017

Índice

1. Descripción de la práctica

Objetivo: El alumno deberá implementar un analizador lexico en Flex para el lenguaje Simplon, creado para la ocasión. El analizador recibirá como argumento el path del fichero de entrada conteniendo el programa que se quiera analizar, y escribirá en la consola (o en un fichero) la lista de tokens encontrados en el fichero de entrada, saltando los comentarios.

Documentación a presentar: El código fuente se enviará a través de Faitic. El nombre del fichero estará formado por los apellidos de los autores en orden alfabético, separados por un guión, y sin acentos ni eñes.

Ej.- DarribaBilbao-DovalMosquera.l

Grupos: Se podrá realizar individualmente o en grupos de dos personas.

Defensa: Consistirá en una demo al profesor, que calificará tanto los resultados como las respuestas a las preguntas que realice acerca de la implementación de la práctica.

Fecha de entrega y defensa: El plazo límite para subirla a Faitic es el 21 de marzo a las 10:30. La defensa tendrá lugar la semana del 20 de marzo en horario de prácticas.

Nota máxima: 1 pto. Se evaluará al alumno por las partes del analizador que se hayan hecho satisfactoriamente:

- 0'125 por las palabras reservadas e identificadores
- 0'125 por los delimitadores y operadores
- 0'25 puntos por las constantes numéricas (enteras y reales)
- 0'25 puntos por constantes caracter, cadenas y archivos de cabecera.
- 0'25 puntos por los comentarios.

Ejemplo: Para un programa como el siguiente:

```
modulo <<mates.sim>>
MODULO <<e-s_std.sim>>

funcion Radio_Circunferencia es

comienzo
   %% Constante
   PI: constante real := .3141592E1;

   %% Variables
   area, radio: real;

   escribir_consola("%nRadio de la {circunferencia}: ");
   leer_consola(radio); { Entrada de dato }

   %% Calculo del area
   area := PI * radio ** 2;

   { El resultado del area se saca por la "consola":
     se trata de un numero real }
   escribir_consola("%nArea de la %
                    %"circunferencia%": \f", area); escribir_consola("%n");

   DeVoLVeR area;
fin
```

la salida debe parecerse a:

```
  linea 1, palabra reservada: modulo
  linea 1, modulo: <<mates.sim>>
  linea 2, palabra reservada: MODULO
  linea 2, modulo: <<e-s_std.sim>>
  linea 4, palabra reservada: funcion
  linea 4, identificador: Radio_Circunferencia
  linea 4, palabra reservada: es
  linea 6, palabra reservada: comienzo
  linea 8, identificador: PI
  linea 8, delimitador: :
  linea 8, palabra reservada: constante
  linea 8, palabra reservada: real
  linea 8, operador :=
  linea 8, ctc real: .3141592E1
  linea 8, delimitador: ;
  linea 11, identificador: area
  linea 11, delimitador: ,
  linea 11, identificador: radio
  linea 11, delimitador: :
  linea 11, palabra reservada: real
  linea 11, delimitador: ;
  linea 13, identificador: escribir_consola
  linea 13, delimitador: (
  linea 13, cadena: "%nRadio de la {circunferencia}: "
  linea 13, delimitador: )
  linea 13, delimitador: ;
```

```
linea 14, identificador: leer_consola
linea 14, delimitador: (
linea 14, identificador: radio
linea 14, delimitador: )
linea 14, delimitador: ;
linea 17, identificador: area
linea 17, operador :=
linea 17, identificador: PI
linea 17, operador *
linea 17, identificador: radio
linea 17, operador **
linea 17, ctc entera: 2
linea 17, delimitador: ;
linea 21, identificador: escribir_consola
linea 21, delimitador: (
linea 21, cadena: "%nArea de la %
                   %"circunferencia%": \f"
linea 22, delimitador: ,
linea 22, identificador: area
linea 22, delimitador: )
linea 22, delimitador: ;
linea 22, identificador: escribir_consola
linea 22, delimitador: (
linea 22, cadena: "%n"
linea 22, delimitador: )
linea 22, delimitador: ;
linea 24, palabra reservada: DeVoLVeR
linea 24, identificador: area
linea 24, delimitador: ;
linea 25, palabra reservada: fin
```

2. Especificación léxica de Simplon

Para que podais escribir el analizador léxico, vamos a especificar a continuación cada uno de los constituyentes léxicos de los programas Simplon.

2.1. Palabras reservadas

Son las siguientes:

```
booleano  bucle  caracter  casos  comienzo  constante  cuando  de  devolver  en  entero
entonces  es  exportar  declaracion  descendente  fin  funcion  mientras  modulo  otro
para  procedimiento rango  real  referencia  registro  repetir  salir  si  sino  subtipo
tabla  tipo  vacia  valor
```

Las palabras reservadas pueden escribirse totalmente en mayúsculas o minúsculas, o en cualquier combinación de ambas.

2.2. Identificadores

Un identificador es una secuencia de caracteres, que pueden pertenecer a las siguientes categorías:

- letras mayúsculas o minúsculas pertenecientes al juego de caracteres ASCII.
- el subrayado: '_'

- dígitos entre '0' y '9'.

Importante: El primer carácter del identificador sólo puede ser una letra o '_'.

Ejemplo:

```
identificadores     NO son identificadores
---------------     ----------------------
uno                 úno
_25diciembre        25diciembre
tabla_123
TABLA
Array_Modificado
```

2.3. Constantes

Vamos a considerar cinco tipos de constantes: números enteros, números reales, caracteres, cadenas y nombres de archivos de cabecera.

Constantes enteras: vamos a considerar tres notaciones: decimal, octal y hexadecimal. En los tres casos las constantes están formadas por uno o más caracteres en los siguientes rangos:

- En notación decimal, los dígitos del '0' al '9'.
- En notación octal, dígitos de '0' a '7'. Además, la secuencia de dígitos estará precedida por un '%o' o '%O'.
- En hexadecimal, los dígitos de '0' a '9' y las letras de la 'a' a la 'f', tanto en mayúcula como en minúscula, con la secuencia '%x' (o '%X') al principio de la constante entera.

Ejemplos:

```
%x23    %% 35 en hexadecimal
%057    %% 47 en octal
%XFfF   %% 4095 en hexadecimal
%023    %% 18 en octal
25
38
```

Constantes reales: consideraremos dos tipos de números decimales:

- Los formados por una parte entera (que es opcional), el punto decimal '.', y una parte fraccionaria. Los dígitos de la parte entera y fraccionaria deben tener la misma codificación: decimal, octal o hexadecimal. En los dos últimos casos, el número estará precedido de la secuencia '%o' y '%x', respectivamente (donde 'o' y 'x' pueden estar en minúscula o mayúscula).

  Ejemplos:

  ```
  .45  38.25 %XF.0a %o.523
  ```

- Los formados por una mantisa y un exponente. La mantisa puede ser un número entero o fraccionario en codificación decimal, octal o hexadecimal. El exponente está formado por el carácter 'e' (o 'E') seguido por uno o más dígitos decimales (entre '0' y '9'), que pueden, opcionalmente, estar precedidos de un signo ('+' o '-').

  Ejemplos:

  ```
  27.5e-7  45E10  .72e+1  %xF8E-4  %0.258e2
  ```

Caracteres: están formadas por dos comillas simples ('), que irán antes y después de:

- un único carácter, excepto el salto de línea, la comilla simple o la secuencia de escape '`%`'.
- los siguientes caracteres escapados:

  `%'  %"  %%  %n  %r  %t`

- el número del caracter expresado en decimal, entre 0 y 255. Cualquer número de 3 dígitos mayor que 255 **no** es un carácter válido.
- el número del carácter expresado en octal, formado por '`%o`' (o '`%O`'), seguido de entre uno y tres dígitos octales. El mayor valor de un caracter en octal es '`%o377`' (=255). Cualquier octal de tres dígitos mayor que '`%o377`' **no** es un carácter válido.
- el número del carácter expresado en hexadecimal, formado por '`%x`' (o '`%X`') y uno o dos dígitos hexadecimales.

Ejemplos:

```
'80'  'a'  '9'  '%n'  '%xD'  '%O116'  '%''
```

Cadenas: están formadas por dos comillas dobles ("), que irán antes y después de una secuencia de 0 o más:

- caracteres, exceptuando el salto de línea, la comilla doble y la secuencia de escape '`%`'.
- los caracteres escapados definidos en el apartado anterior.
- '`%`', seguido de un salto de línea.

Ejemplos:

```
"%nRadio de la {circunferencia}: "    %% es una cadena
"primera %                            %% es una cadena escrita
 segunda %                            %% en tres lineas
 tercera"
```

Nombres de módulos: están formados por '`<<`' y '`>>`', delimitando una secuencia de uno o más caracteres excepto el salto de línea, '`<`' y '`>`'.

Ejemplo:

```
modulo <</home/fulanito/carpeta/nombre.sim>>
```

2.4. Delimitadores

Consideramos los siguientes (no los he entrecomillado para facilitar la lectura):

```
(   )   :   ;   ,   ..   |   =>
```

2.5. Operadores

Consideramos los siguientes clases:

- Asignación

  `:=`

- Operadores aritméticos:

```
+   -   *   /   \ (modulo)   **   --   ++
```

- Operadores de acceso a memoria (estructuras y tablas):

```
.   [   ]
```

- Operadores de bits:

```
<- (desplazamiento izda)   -> (desplazamiento dcha)
```

- Operadores relacionales:

```
<   >   <=   >=   =   <>
```

- Operadores lógicos:

```
~ (negacion)   /\ (and)   \/ (or)
```

2.6. Comentarios

En Simplon podemos encontrar dos tipos de comentarios:

- los que comienzan con la secuencia '%%' y abarcan hasta el final de la línea.
- los comentarios multilínea, delimitados por '{' y '}'.

Importante: Cuando las secuencias '%%', '{' o '}' aparecen dentro de un cadena, el nombre de archivo de cabecera o un comentario **no** pueden ser interpretados como comentarios.

Importante (otra vez): Se ignorará la posibilidad de anidamiento de comentarios multilínea. Se considera que si, en el futuro, algún programador siente la tentación de anidar comentarios multilínea en su código, será detectado y puesto bajo la custodia de la sección Q del SOE antes de que pueda hacerlo.

Ejemplos:

```
"a%%b"                      %% una cadena
modulo <<%%e>>              %% un nombre de archivo de cabecera
%% }                        %% un comentario de una sola línea
f := g{}/h;                 %% f := g / h;
m := n%%*}o
    + p;                    %% m := n + p;
```

2.7. Errores

Cuando el analizador encuentre una porción de la cadena de entrada que no se corresponda con ninguno de los tokens anteriores (o con espacios, tabuladores o saltos de línea), devolverá un mensaje de error, indicando la línea en el que ha encontrado el error. Sin embargo, el análisis proseguirá hasta agotar el fichero de entrada.