Practica 1.- Analizador Léxico de Micro

TALF 20-21 (Julio)

15 de junio de 2021

Índice

1. Descripción de la práctica

Objetivo: El alumno deberá implementar un analizador lexico en Flex para el lenguaje Micro, 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 de Flex con la especificación del analizador léxico se subirá a Moovi. El nombre del fichero estará formado por los apellidos del autor (o autores) en orden alfabético, sin acentos ni eñes. **Sólo se subirá el archivo fuente de Flex**.

Ej.- DarribaBilbao-VilaresFerro.l

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

Fecha de entrega: El plazo límite para subirla a Moovi es el mismo día del examen, 2 de julio de 2021, a las 23:59.

Material: He dejado en Moovi (TALF → Documentos y Enlaces → Material de prácticas → Práctica 1) un directorio comprimido, `micro.flex.tar.gz`, con los siguientes archivos:

- `micro.l`, donde podeis escribir vuestras especificaciones.
- `prueba1.mi` y `prueba2.mi`, archivos con código Micro para probar el analizador.
- `tokens.h`, con las macros en C en las que se especifican los nombres de los tokens.
- `Makefile`, para compilar la especificación Flex y generar un ejecutable, de nombre `micro`.

Nota máxima: 1'5 ptos. Se evaluará al alumno por las partes del analizador que se hayan hecho satisfactoriamente:

- 0'2 ptos por las palabras reservadas e identificadores.
- 0'3 ptos por los delimitadores y operadores.
- 0'4 ptos por las constantes numéricas (enteras y reales).
- 0'4 ptos por constantes caracter, cadenas y valores booleanos.
- 0'2 ptos por los comentarios y errores.

2. Acciones y salida del analizador

A diferencia de lo que hacíamos en años anteriores, en lugar de analizar la entrada con una única llamada a `yylex()`, el analizador devolverá el control del programa cada vez que encuentre una categoría léxica (token) de Micro. Por lo tanto, las acciones constarán de al menos dos instrucciones: una para la salida por la consola o fichero (por ejemplo, `printf()` o `fprintf()`) y un `return`, devolviendo el nombre de la constante definida en `tokens.h` (o `micro.tab.h`) correspondiente al token encontrado.

Por lo tanto, no será suficiente hacer una sola llamada a `yylex()` en el código, sino que habrá que seguir llamando al analizador hasta agotar la entrada. La forma más fácil de hacerlo es dentro de un bucle.

```
while (yylex());
```

El objetivo de este cambio es hacer más fácil la reutilización del analizador léxico que vais a implementar en la práctica 2.

Con respecto a los nombres de las constantes que se devolverán cada vez que se encuentre un token, tendremos los siguientes casos:

- Para las palabras reservadas, los operadores '`mod`' y '`and`', '`or`' y '`not`', y las constantes '`true`' y '`false`' el nombre que se devolverá será el token encontrado en mayúsculas. Por ejemplo, cuando se detecte la palabra reservada '`exit`', se ejecutará `return(EXIT);`
- Para los tokens que consten de un sólo carácter, se devolverá el código ASCII de dicho carácter: `yytext[0]`.
- El token correspondiente a un identificador es `IDENTIFICADOR`.
- Para los valores constantes, se devolverán los tokens `CTC_INT` (números enteros), `CTC_FLOAT` (números reales), `CTC_CADENA` y `CTC_CARACTER`.
- Para el resto de categorías léxicas, listamos a continuación sus nombres:

```
'->'  FLECHA        '..'  DOS_PTOS       ':='  ASIG            '**'  EXP
'/='  DISTINTO      '>='  MAYOR_IGUAL    '<='  MENOR_IGUAL
```

Ejemplo: Para un código como el siguiente:

```
procedure ORDENAR_POR_MONTONES is
    N: constant INTEGER := 10;
    TABLA: array(1..N) of INTEGER;
    TEMPORAL: INTEGER;

    procedure METER(I,N: in out INTEGER) is
        TEMPORAL,J: INTEGER;
        TERMINAR: BOOLEAN;
    start
        TEMPORAL := TABLA(I);

...
```

la salida debe parecerse a:

```
Linea 1 - Palabra Reservada: procedure
Linea 1 - Identificador: ORDENAR_POR_MONTONES
Linea 1 - Palabra Reservada: is
Linea 2 - Identificador: N
Linea 2 - Delimitador: :
Linea 2 - Palabra Reservada: constant
Linea 2 - Palabra Reservada: INTEGER
Linea 2 - Operador: :=
Linea 2 - Entero: 10
Linea 2 - Delimitador: ;
Linea 3 - Identificador: TABLA
Linea 3 - Delimitador: :
Linea 3 - Palabra Reservada: array
Linea 3 - Delimitador: (
Linea 3 - Entero: 1
Linea 3 - Delimitador: ..
Linea 3 - Identificador: N
Linea 3 - Delimitador: )
Linea 3 - Palabra Reservada: of
Linea 3 - Palabra Reservada: INTEGER
Linea 3 - Delimitador: ;
Linea 4 - Identificador: TEMPORAL
Linea 4 - Delimitador: :
Linea 4 - Palabra Reservada: INTEGER
Linea 4 - Delimitador: ;
Linea 6 - Palabra Reservada: procedure
Linea 6 - Identificador: METER
Linea 6 - Delimitador: (
Linea 6 - Identificador: I
Linea 6 - Delimitador: ,
Linea 6 - Identificador: N
Linea 6 - Delimitador: :
Linea 6 - Palabra Reservada: in
Linea 6 - Palabra Reservada: out
Linea 6 - Palabra Reservada: INTEGER
Linea 6 - Delimitador: )
Linea 6 - Palabra Reservada: is
Linea 7 - Identificador: TEMPORAL
Linea 7 - Delimitador: ,
Linea 7 - Identificador: J
Linea 7 - Delimitador: :
Linea 7 - Palabra Reservada: INTEGER
Linea 7 - Delimitador: ;
Linea 8 - Identificador: TERMINAR
Linea 8 - Delimitador: :
Linea 8 - Palabra Reservada: BOOLEAN
Linea 8 - Delimitador: ;
Linea 9 - Palabra Reservada: start

...
```

3. Especificación léxica de Micro

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

3.1. Palabras reservadas

```
array  boolean  case  character  constant  else  exit  false  finish  float  for  foreach
function  hashtable  if  in  integer  is  loop  nil  of  others  out  procedure  record
return  reverse  start  then  true  type  start  when  while
```

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

3.2. Identificadores

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

- letras mayúsculas o minúsculas (sin acentos de ningún tipo) pertenecientes al juego de caracteres ASCII.
- el guión bajo: '_'
- dígitos entre '0' y '9'.

Importante: El primer carácter del identificador sólo puede ser una letra. Si en el resto de la secuencia aparecen uno o más digitos decimales consecutivos, el primero de ellos tiene que estar precedido por '_'.

Ejemplo:

```
identificadores                              NO son identificadores
---------------                              ----------------------
uno     el_25diciembre     tabla_123         úno   _25diciembre  tabla123
TABLA   Array_Modificado   hola__25
```

3.3. Constantes

Vamos a considerar cinco tipos de constantes: números enteros, números reales, caracteres, cadenas y valores booleanos.

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 la secuencia '\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 '\h' (o '\H') al principio de la constante entera.

Ejemplos:

```
\h23    ## 35 en hexadecimal
\O057   ## 47 en octal
\HFfF   ## 4095 en hexadecimal
\o23    ## 18 en octal
25
38
```

Constantes reales: consideraremos dos tipos de números reales:

- Los formados por una parte entera, el punto decimal '`.`', y una parte fraccionaria. Los dígitos de la parte entera y fraccionaria pueden tener distintas codificaciones: decimal, octal o hexadecimal. En los dos últimos casos, la secuencia de dígitos estará precedida de la secuencia '`\o`' y '`\h`' (o '`\O`' '`\H`'), respectivamente.

  Ejemplos:

  ```
  0.45  38.25  \hF.\O066  1.\o523  76.\HAF
  ```

- Los formados por una mantisa y un exponente. La mantisa puede ser un número entero o fraccionario en codificación decimal, octal o hexadecimal (en este último caso las codificaciones de la parte entera y fraccionaria pueden ser distintas). El exponente está formado por el carácter '`e`' (o '`E`') seguido por un número entero (octal, decimal o hexadecimal), que puede, opcionalmente, estar precedido de un signo ('`+`' o '`-`').

  Ejemplos:

  ```
  \o27.5e-\h7A  45e10  0.\O72e+10  \HF8E-\Ha4  22.254e2
  ```

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:

  ```
  \'  \"  \\  \a  \b  \f  \n  \r  \t  \v
  ```

Ejemplos:

```
'\n'  'a'  '9'  '\\'  '#'
```

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

- caracteres, excepto el salto de línea, la comilla doble o la secuencia de escape '`\`'.
- los caracteres escapados enumerados anteriormente.
- la secuencia de escape '`\`', seguida de un salto de línea.

Ejemplos:

```
"hola"  "adios\n"  "uno\tdos\ttres"  "999"
 -- la siguiente cadena abarca tres líneas
"primera \
 segunda \
 tercera"
```

Valores booleanos: Son las constantes '`True`' y '`False`'. Al igual que las palabras reservadas y los operadores '`and`', '`or`' y '`not`', se podrán escribir en mayúsculas, minúsculas o cualquier combinación de ambas.

3.4. Delimitadores

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

```
{   }   (   )   [   ]   |   :   ;   ,   ->   ..
```

3.5. Operadores

Consideramos los siguientes clases de operadores:

- Aritméticos (los dos últimos son, respectivamente, la división y la potencia):

  ```
  +   -   *   %   **
  ```

  ('-' está sobrecargado como el operador de cambio de signo y la resta)

- De bits (or y and):

  ```
  @   &
  ```

- Relacionales (el segundo es el operador distinto):

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

- Lógicos (se pueden escribir en mayúsculas, minúsculas o cualquier combinación de ambas):

  ```
  not   and   or
  ```

- De asignación:

  ```
  :=
  ```

- De acceso a memoria (a campos dentro de un registro):

  ```
  .
  ```

A la hora de escribir en la consola la cadena indicando el reconocimiento de un operador, no es necesario que escribais el tipo de operador (aritmético, de bits, asignación, etc), pero podeis hacerlo si quereis.

3.6. Comentarios

En Micro sólo vamos a tener comentarios de una sola línea, que empiezan por la secuencia '--'. Los comentarios serán ignorados por el analizador léxico, no generando salida alguna.

Importante: Cuando la secuencia '--', aparece dentro de una cadena **no** puede ser interpretada como el comienzo de un comentario.

Ejemplos:

```
'a--b'                    -- una cadena
-- comentario             -- un comentario de una sola línea
```

3.7. 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.