Practica 1.- Analizador Léxico de Morónico

TALF 2022/23

22 de marzo de 2023

Índice

1. Descripción de la práctica

Objetivo: El alumno deberá implementar un analizador lexico en Flex para el lenguaje Morónico, 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-VilaresFerro.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 24 de marzo a las 23:59. La defensa tendrá lugar la semana del 27 de marzo en horario de prácticas.

Material: Hemos dejado en Faitic (TALF → documentos y enlaces → material de prácticas → Práctica 1) un fichero (`moronico.tar.gz`). Dicho fichero contiene la especificación incompleta[1] en Flex del analizador léxico (`moronico.l`), un fichero con la definición de los nombres de los tokens de la especificación (`moronico.h`), y un archivo para probar el analizador (`prueba.mor`).

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

[1] En realidad, prácticamente vacía.

- 0'25 por las palabras reservadas e identificadores
- 0'25 por los delimitadores y operadores
- 0'25 puntos por las constantes numéricas (enteras y reales)
- 0'5 puntos por constantes caracter, cadenas y archivos de cabecera.
- 0'25 puntos por los comentarios.

2. Acciones y salida del analizador

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 Morónico. 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 `moronico.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 de este analizador léxico 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 los tokens formados por un único carácter, se devuelve el propio carácter.
- Para las palabras reservadas, el nombre de token será la palabra en mayúsculas. Por ejemplo, para '`real`', el nombre de token correspondiente será `REAL`, para '`interfaz`' será `INTERFAZ`, etc.
- Los nombres del resto de categorías léxicas (junto con el token en cuestión entre comillas) los teneis en la siguiente lista:

```
AND '/\'                    CTC_BOOLEANA 'verdadero' o 'falso'  CTC_CADENA (una cadena)
CTC_CARACTER (un carácter)  CTC_ENTERA (núm. entero)            CTC_REAL (núm. real)
CUATRO_PTOS '::'            DESPD '->'                          DESPI '<-'
DOS_PTOS '..'               EQ ':='                             FLECHA_DOBLE '=>'
GEQ '>='                    IDENTIFICADOR (un identificador)    LEQ '=<'
NEQ '!='                    OR '\/'                             PATH (un path)
POTENCIA '**'
```

Ejemplo: Para un código como el siguiente:

```
programa prueba;

tipo

  FECHA = registro {
    DIA : entero;
    MES : entero;
    ANNO : entero;
    }

  dia_semana = ("lunes","martes","miércoles","jueves","viernes");
```

```
  t_hash_cadenas = lista asociativa de cadena;

constante

  PI : real = 3'141592;  // constante del programa

variable

  HOY : dia_semana;

  /* esto es un
     comentario multilinea
     pero que sigue en otra linea */

  lista_int : lista [1..100] de entero = [ 1, \x7AF, \25, 400];
  acumulador : entero;
  mi_hash : lista asociativa de entero;

  lista_cadenas : t_hash_cadenas = [ "uno"  => "prueba\
                                      ",
                                     "dos"  => "fantastico",
                                     "tres" => "\nRadio de la /*circunferencia*/\
                                                \151\144\X69\157\x74\141: "];

  lista_real : lista [1..100] de real = [ '45^-27, 5'21^18, 91'01];

{
  HOY = "martes";

  en caso HOY sea
    "lunes" .. "jueves"  => { trabajo(); }
    "viernes" | "sabado" => { trabajo(); deporte(); }
     otro                => { ; };

  valor = mi_hash[hoy];   // lectura de un valor de una lista asociativa
  imprimir(mi_hash[hoy]);

  mi_hash[manana] = 123;  // escritura de un valor (manana es una variable de tipo FECHA)

  acumulador = 0;
  para elemento_int en lista_int {
    acumulador = acumulador + elemento_int;
    }
}
```

la salida debe parecerse a:

```
linea 1, palabra reservada: programa
linea 1, identificador: prueba
linea 1, delimitador: ;
linea 3, palabra reservada: tipo
linea 5, identificador: FECHA
linea 5, operador: =
linea 5, palabra reservada: registro
linea 5, delimitador: {
linea 6, identificador: DIA
linea 6, delimitador: :
```

```
linea 6, palabra reservada: entero
linea 6, delimitador: ;
linea 7, identificador: MES
linea 7, delimitador: :
linea 7, palabra reservada: entero
linea 7, delimitador: ;
linea 8, identificador: ANNO
linea 8, delimitador: :
linea 8, palabra reservada: entero
linea 8, delimitador: ;
linea 9, delimitador: }

...

linea 50, operador: [
linea 50, identificador: manana
linea 50, operador: ]
linea 50, operador: =
linea 50, constante entera: 123
linea 50, delimitador: ;
linea 52, identificador: acumulador
linea 52, operador: =
linea 52, constante entera: 0
linea 52, delimitador: ;
linea 53, palabra reservada: para
linea 53, identificador: elemento_int
linea 53, palabra reservada: en
linea 53, identificador: lista_int
linea 53, delimitador: {
linea 54, identificador: acumulador
linea 54, operador: =
linea 54, identificador: acumulador
linea 54, operador: +
linea 54, identificador: elemento_int
linea 54, delimitador: ;
linea 55, delimitador: }
linea 56, delimitador: }
```

3. Especificación léxica de Morónico

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

3.1. Palabras reservadas

Son las siguientes:

```
abstracto asociativa booleano cabecera cadena caso caracter clase conjunto constante cuerpo
constructor cuando descendente destructor de devolver carga elemento ejecuta en entero
entonces especifico excepto fichero final finalmente funcion generico hasta interfaz
lanzar lista mientras modificable otro paquete para privado probar procedimiento programa
publico real registro repite salir sea semipublico si sino tipo variable
```

Las palabras reservadas pueden escribirse totalmente en mayúsculas o minúsculas, o en cualquier combinación de ambas. Hay un modificador en la sintaxis de expresiones regulares de flex que permite hacer a una expresión regular

insensible a mayúsculas y minúsculas. Lo teneis en la versión en inglés del manual.

3.2. 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
```

3.3. Constantes

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

Constantes enteras: este tipo de constantes están formadas por:

- uno o más caracteres en el rango de dígitos del '0' al '9' (codificación decimal).
- uno o más caracteres en el rango de dígitos del '0' al '7' (codificación octal), precedidos del símbolo de escape '\'.
- uno o más caracteres en los rangos '0' a '9' y 'a' a 'f', tanto en mayúsculas como monúsculas (codificación hexadecimal), y precedidos de '\x' o '\X'

Ejemplos:

```
0  \25  099  \x1aF  \XFFF
```

Constantes reales: consideraremos dos tipos de números reales **decimales**:

- Los formados por una parte entera (que es opcional), la coma decimal '', y una parte fraccionaria.
  Ejemplos:

  ```
  '45  38'25432
  ```

- Los formados por una mantisa y un exponente **decimales**. La mantisa puede ser un número entero o fraccionario. El exponente está formado por el carácter '^' seguido por uno o más dígitos decimales, que pueden, opcionalmente, estar precedidos de un signo ('+' o '-').
  Ejemplos:

  ```
  27'5^-7  45^10  '72^+1
  ```

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

- un único carácter, excepto el salto de línea, la comilla simple, la comilla doble, la interrogación, o la secuencia de escape '\'.
- los siguientes caracteres escapados (ahora incluye `\"`):

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

- el número del caracter expresado en decimal, entre 0 y 255. Cualquier número de 3 dígitos mayor que 255 **no** es un carácter válido.
- el número del carácter expresado en octal, formado por '\', seguido de entre uno y tres dígitos octales (de `0` a `7`). El máximo valor de un caracter en octal es '\377' (=255). Cualquier octal de tres dígitos mayor **no** es un carácter válido.
- el número del carácter expresado en hexadecimal, formado por '\`x`' y uno o dos dígitos hexadecimales (de `0` a `9` y de `a` a `f`, tanto en mayúsculas como en minúsculas).

Ejemplos:

```
'\0'  'a'  '9'  '\n'  '\xD'  '\xaF'  '\''
```

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

- los mismos caracteres no escapados que en el caso del token carácter.
- los caracteres escapados definidos en el apartado anterior.
- '\', seguido de un salto de línea.

Ejemplos:

```
"\nRadio de la /*circunferencia*/: "  // es una cadena
"linea1 \                             // es una cadena
 linea2 \                             // que abarca más de una línea
 linea3"
"eres un \151\144\x69\157\x74\141\n"  // es una cadena
```

Paths: están formados por '`"`', delimitando una secuencia de uno o más caracteres excepto el salto de línea y '`"`', que comienzan por '`/`', '`./`' o '`/../`'.

Ejemplo:

```
"./modulo.mor" "/home/victor/path/modulo.mor" "../path/modulo.mor"
```

En el caso de que una secuencia de caracteres pueda ser reconocida como cadena y path, el analizador optará por el token path.

3.4. Delimitadores

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

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

3.5. Operadores

Consideramos los siguientes clases:

- Asignación

  ```
  =
  ```

- Operadores aritméticos:

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

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

  ```
  .   [   ]
  ```

- Operadores de bits:

  ```
  <- (desplazamiento izda)   -> (desplazamiento dcha)
  @ (or binario)   ~ (xor binario)   & (and binario)
  ```

- Operadores relacionales:

  ```
  <   >   =<   >=   :=   !=
  ```

- Operadores lógicos:

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

3.6. Comentarios

En Morónico 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
".//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;
```

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.