Practica 1.- Analizador Léxico de Chusco

ALF 23-24 - Evaluación Contínua

6 de marzo de 2024

Índice

1. Descripción de la práctica

Objetivo: El alumno deberá implementar un analizador lexico en Flex para el lenguaje Chusco, 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 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. **Sólo se subirá el archivo fuente de Flex**.

Ej.- DarribaBilbao-OuteirinoCid.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 2 de abril a las 23:59. La defensa tendrá lugar en las clases de prácticas de los días 3 y 5 de abril.

Material: He dejado en Faitic (ALF → Documentos e Ligazóns → Material de prácticas) un directorio comprimido, `chusco.flex.tar.gz`, con los siguientes archivos:

- `chusco.l`, donde podeis escribir vuestra especificación.
- `prueba.chu`, para probar el analizador léxico resultante.
- `Makefile`, para compilar vuestra especificación Flex y generar un ejecutable, de nombre `chusco`.

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

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

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 Chusco. 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 `chusco.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 '`bucle`', el nombre de token correspondiente será `BUCLE`, para '`entero`' será `ENTERO`, etc.
- `IDENTIFICADOR` corresponde a un identificador.
- `CTC_CARACTER` corresponde a un carácter (incluyendo las comillas simples).
- `CTC_CADENA` corresponde a una cadena (incluyendo las comillas dobles).
- `CTC_ENTERA` corresponde a un número entero.
- `CTC_REAL` corresponde a un número real.
- Los nombres para el resto se símbolos de más de un carácter son:

```
'..' DOS_PUNTOS   '::' CUATRO_PUNTOS   ':=' ASIGNACION   '=>' FLECHA     '++' INC
'--' DEC          '<-' DESPI           '->' DESPD        '<=' LEQ        '>=' GEQ
'~=' NEQ          '/\' AND             '\/' OR           ':+' ASIG_SUMA  ':-' ASIG_RESTA
':*' ASIG_MULT    ':/' ASIG_DIV        ':\' ASIG_RESTO   ':^' ASIG_POT   ':<' ASIG_DESPI
':>' ASIG_DESPD
```

Ejemplo: Para un programa como el siguiente:

```
subprograma Radio_Circunferencia

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

   ## Variables
   area, radio: real;
   otra_cosa : real := %x1F.34^-Faa;

   escribir_consola("%nRadio de la #{circunferencia#}%
                    %o151%O144%X69%O157%x74%O141: ");
   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 subprograma
```

la salida debe parecerse a:

```
linea 1, palabra reservada: programa
linea 1, identificador: Radio_Circunferencia
linea 1, delimitador: ;
linea 3, palabra reservada: principio
linea 5, identificador: PI
linea 5, delimitador: :
linea 5, palabra reservada: constante
linea 5, palabra reservada: real
linea 5, operador: :=
linea 5, ctc real: .3141592^1
linea 5, delimitador: ;
linea 8, identificador: area
linea 8, delimitador: ,
linea 8, identificador: radio
linea 8, delimitador: :
linea 8, palabra reservada: real
linea 8, delimitador: ;
linea 9, identificador: otra_cosa
linea 9, delimitador: :
linea 9, palabra reservada: real
linea 9, operador: :=
linea 9, ctc real: %x1F.34^-Faa
linea 9, delimitador: ;
linea 11, identificador: escribir_consola
linea 11, delimitador: (
linea 11, cadena: "%nRadio de la #{circunferencia#}%
                    %o151%O144%X69%O157%x74%O141: "
linea 12, delimitador: )
```

```
linea 12, delimitador: ;
linea 13, identificador: leer_consola
linea 13, delimitador: (
linea 13, identificador: radio
linea 13, delimitador: )
linea 13, delimitador: ;
linea 16, identificador: area
linea 16, operador: :=
linea 16, identificador: PI
linea 16, operador: *
linea 16, identificador: radio
linea 16, operador: ^
linea 16, ctc entera: 2
linea 16, delimitador: ;
linea 20, identificador: escribir_consola
linea 20, delimitador: (
linea 20, cadena: "%nArea de la %
                    %"circunferencia%": #f"
linea 21, delimitador: ,
linea 21, identificador: area
linea 21, delimitador: )
linea 21, delimitador: ;
linea 21, identificador: escribir_consola
linea 21, delimitador: (
linea 21, cadena: "%n"
linea 21, delimitador: )
linea 21, delimitador: ;
linea 23, palabra reservada: DeVoLVeR
linea 23, identificador: area
linea 23, delimitador: ;
linea 24, palabra reservada: fin
```

3. Especificación léxica de Chusco

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

3.1. Palabras reservadas

```
abstracto booleano bucle caracter casos clase como constante constructor corto cuando de
descendente destructor devolver diccionario en entero entonces enumeracion es especifico
excepcion exportar falso fin final finalmente generico importar largo lanza libreria lista
mientras objeto otro para principio privado programa protegido prueba publico rango real
referencia registro repetir salir si signo siguiente sino subprograma tabla tipo ultima
valor verdadero
```

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 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 cuatro tipos de constantes: números enteros, números reales, caracteres y cadenas.

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 reales:

- 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 '^' seguido por uno o más dígitos con la misma codificación que la mantisa (octal, decimal o hexadecimal), que pueden, opcionalmente, estar precedidos de un signo ('+' o '-').

  Ejemplos:

```
%027.5^-7  45^10  .72^+1  %xF8^-a4  %0.254^2
```

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 carácter expresado en decimal, entre 0 y 255, precedido de '%'. Un 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:

```
'%o171'  'a'  '%255' '9'  '%n'  '%xD'  '%X1f'  '%''
```

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, las comillas dobles y la secuencia de escape '%'.
- los caracteres escapados definidos en el apartado anterior.
- los caracteres en decimal, octal o hexadecimal 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"
```

3.4. Delimitadores

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

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

3.5. Operadores

Consideramos los siguientes clases:

- Operadores aritméticos (los dos últimos son, respectivamente el módulo y la potencia):

  ```
  +   -   *   /   --   ++   \   ^
  ```

- Operadores de bits (desplazamiento izda y desplazamiento dcha):

  ```
  <-   ->
  ```

- Asignación (el primero es la asignación a secas y el resto aplican uno de los operadores aritméticos o de bits sobre la variable en la que se almacena el resultado)

  ```
  :=   :+   :-   :/   :\   :^   :<   :>
  ```

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

  ```
  .   [   ]   {   }   ::
  ```

- Operadores relacionales:

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

- Operadores lógicos (respectivamente, negación, and y or):

  ```
  ~   /\   \/
  ```

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 Chusco 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 o un comentario **no** pueden ser interpretados como comentarios.

También Importante: 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
## #}                     ## 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.