Analizador Léxico

Garay, Iñaki LU 67387; Touceda, Tomás LU 84024

30 de agosto de 2011

Analizador Léxico
by Garay, Iñaki LU 67387; Touceda, Tomás LU 84024

Índice general

Capítulo 1

Especificación e instrucciones de uso

Existen dos maneras de correr el analizador léxico sobre un archivo de entrada:

1. Utilizando el archivo ejecutable lexor.exe provisto:

   ```
   C:\> lexor.exe <archivo_entrada> [<archivo_salida>]
   ```

2. Utilizando el intérprete de Python para correr el script del módulo principal:

   ```
   C:\> python.exe lexor_main.py <archivo_entrada> [<archivo_salida>]
   ```

La especificación del archivo de salida es opcional. Si se omite, el analizador léxico mostrará su salida por pantalla. Si no se especifica un archivo de entrada, o se pasan más argumentos que el archivo de entrada y el de salida, el analizador léxico mostrará un mensaje explicando el uso por linea de comando.

1.1. Formato de salida

Mientras procesa el archivo, el analizador léxico mostrará en su salida los tokens reconocidos en una tabla, con el siguiente formato:

```
$ python2 lexor_main.py ../tests/lexor/003err-ampersand.java
1:0     -   <INT_TYPE>      :: int
1:3     -   <SEPARATOR>     ::
1:4     -   <IDENTIFIER>    :: fun
1:7     -   <PAREN_OPEN>    :: (
1:8     -   <INT_TYPE>      :: int
1:11    -   <SEPARATOR>     ::
1:12    -   <IDENTIFIER>    :: a
1:13    -   <PAREN_CLOSE>   :: )
1:14    -   <SEPARATOR>     ::
1:15    -   <BRACE_OPEN>    :: {
1:16    -   <SEPARATOR>     ::
2:4     -   <IDENTIFIER>    :: un
2:6     -   <SEPARATOR>     ::
ERROR: Line: 2, Col: 7 :: Token no reconocido.
In line 2:7
    un & amper;
-------^
```

La primera columna de la tabla indica el número de linea y la columna en la cual se encontró el token. La siguiente columna indica el tipo del token. La tercera columna muestra el lexema del token.

1.2. Alfabeto de entrada

El alfabeto de entrada son todos los caracteres de la codificación ASCII.

1.3. Definición de Tokens

Aclaraciones:

1. Para evitar ambiguedades, las expresiones regulares se expresaron utilizando la sintaxis válida para JLex.
2. Las entradas que no cuentan con ejemplos tienen un unico representante, igual a la expresión regular con la cual coincide.

La siguiente tabla muestra los tokens reconocidos por el analizador léxico.

Token	**Expresión regular**	**Ejemplos**
IDENTIFIER	[a-zA-Z_\$][a-zA-Z_\$0-9]*	hola
SEPARATOR	[\t\n\r]	
INT_LITERAL	(0|[1-9]([0-9])*)	42
CHAR_LITERAL	[ver nota *1*]	′q′
STRING_LITERAL	[ver nota *2*]	"hola"
SCOLON	;	;
BRACE_OPEN	{	{
BRACE_CLOSE	}	}
PAREN_OPEN	\)	(
PAREN_CLOSE	\(	)
CLASS	class	class
EXTENDS	extends	extends
PUBLIC	public	public
PROTECTED	protected	protected
STATIC	static	static
THIS	this	this
SUPER	super	super
VOID_TYPE	void	void
BOOLEAN_TYPE	bool	bool
INT_TYPE	int	int
CHAR_TYPE	char	char
IF	if	if
THEN	then	then
ELSE	else	else
WHILE	while	while
RETURN	return	return
TRUE	true	true
FALSE	false	false
NULL	null	null
NEW	new	new
ASSIGNMENT	=	=
CONDITIONAL_AND	&&	&&
CONDITIONAL_OR	||	||
EQUALS	==	==
NOT_EQUALS	!=	!=
LT	<	<
GT	>	>
LT_EQ	< =	< =
GT_EQ	> =	> =
ADD	+	+
SUB	-	-
MUL	*	*
DIV	\/	/
MOD	%	%
NOT	!	!
ACCESSOR	.	.
EOF	<EOF>	

Nota 1 (\'[^('\\)]\')|(\'[\\\\|\\'|\\\"|\\n]\')
Nota 2 (\"\"|\"([^(\"\\)]|[\\\\|\\'|\\\"|\\n])*\")

Capítulo 2

Implementación

La principal decisión de diseño que afecto la implementación fue la abstracción de los estados del automata finito reconocedor.

El analizador léxico se desarrolló utilizando únicamente la versión 2.7 del lenguaje Python (`www.python.org`). El archivo ejecutable para Windows se generó con la herramienta `py2exe` (`www.py2exe.org`).

Para implementar el analizador léxico se realizó la especificación de la máquina de estados basándose en las expresiones regulares definidas en la sección anterior, para luego representarlo en código Python.

2.1. Archivos y clases

2.1.1. Clase `State`

De esta forma, la clase principal utilizada es la llamada State. Esta no es más que una abstracción de la idea es estado de un autómata finito, que cuenta con una serie de funciones de checkeo para determinar si se debe pasar a un siguiente estado o no, y alrededor de esta idea se estableció la lógica para la detección de ciertos errores léxicos.

En el archivo states.py se encuentra la definición de esta clase mencionada, y la del autómata finito reconocedor utilizado.

2.1.2. Clases `Token` y `TokenType`

Para representar a los tokens se armaron dos clases: TokenType y Token. TokenType, definida en constants.py, es una abstracción sobre los distintos tipos de tokens reconocidos y especificados en la sección anterior. Token, definida en lexor.py, abstrae un token instanciado según el análisis del archivo de código fuente en cuestión.

2.1.3. Clase `Lexor`

Por último, Lexor es la clase que representa al analizador léxico propiamente. Este implementa el método get_token() que devuelve secuencialmente todos los tokens reconocidos en un dado archivo que contiene código en MiniJava. Esta clase se encuentra definida en lexor.py.

2.1.4. Clase `LexicalError`

Para el manejo de errores, se creó el tipo de excepción LexicalError dentro del archivo errors.py.

2.1.5. Archivo `lexor_main.py`

El módulo principal del analizador léxico esta implementado en este script.

Capítulo 3

Verificación

3.1. Errores detectados

El analizador léxico reconoce los siguientes tipos de errores:

- Comentario del tipo /* */ que no esté propiamente cerrado.
- Informa de tokens no reconocidos.
- Caracter no reconocido: si se intenta ingresar un caracter que no pertenece al alfabeto se producirá un error.
- Si el archivo de entrada especificado no existe, se producirá un error.
- Si se encuentra un caracter literal de mas de un caracter, e.g. 'hola\', se producirá un error. En cambio, si es un caracter válido, e.g. '\n', se aceptará.

3.2. Casos de prueba

Cuando un error es detectado, también se muestra por pantalla la ubicación de la porción del texto que presenta el error. Por ejemplo:

```
ERROR: Line: 3, Col: 4 :: Comentario no cerrado.
In line 3:4
    /* comment
____^
```

```
ERROR: Line: 2, Col: 10 :: Token no reconocido.
In line 2:10
    a = 1 # 2
__________^
```

- Test 000 Correcto

Descripción: Este es el caso de test principal para detección correcta de tokens.
Resultado esperado: Todos los tokens son reconocidos con éxito según lo esperado, y los comentarios son obviados sin modificar la posición final de los tokens alrededor de ellos.

- Test 001 Erroneo / Token inválido

Descripción: Este caso de test esta destinado a detectar un identificador o palabra clave inválido.
Resultado esperado: Detecta que pub\\lic no es un identificador válido.

- Test 002 Erroneo / Operador inválido

Descripción: La idea de este caso de test es mostrar que se detectan operadores no válidos.
Resultado esperado: Se deberá informar acerca del operador erroneo, asi como mostrar por pantalla la ubicación del mismo.

- Test 003 Erroneo / Operador inválido

Descripción: Algunos operadores contienen más de un carácter, la idea este caso de test es mostrar que se detectan situaciones erroneas como la escritura de "&" en lugar de "&&", siendo este último, el único operador con el caracter '&\' válido.
Resultado esperado: Se deberá detectar el token erróneo, y se mostrará por pantalla la posición del mismo en el archivo de código fuente.

- Test 004 Erroneo / Comentario no finalizado

Descripción: Uno de los problemas con el tipo de comentarios con secuencia de caracteres de apertura y cierre es que puede que no se cierre correctamente. La idea de este caso de test es mostrar que dichas situaciones se detectan.
Resultado esperado: Se detecta el comentario que no ha sido finalizado y se muestra exactamente cual es (en caso de que existan otros).

- Test 005 Correcto / Comentarios bien formados

Descripción: La idea de este test es mostrar que aun con comentarios, los números de linea y columna siguen siendo calculados correctamente. **Resultado esperado:** Los tokens se detectan con éxito y las lineas y columnas se computan de forma correcta.

- Test 006 Erroneao / Literal de caracter invalido.

Descripcion: Los literales de caracteres se especifican con comillas simples. El caracter especificado debe ser un caracter valido, o una secuencia de escape que designe un caracter valido. El proposito de este test es mostrar que un literal de caracter invalido con una sequencia erronea entre las comillas simples es detectado.
Resultado esperado: Se detecta el token invalido.