Proyecto de Autómatas y Lenguajes

Curso 2018-2019

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTONOMA DE MADRID

Práctica 2: ANÁLISIS MORFOLÓGICO

Fecha de entrega según calendario publicado en moodle

Objetivo de la práctica

El objetivo de la práctica es la programación del analizador morfológico del compilador utilizando la herramienta Flex. El analizador morfológico (analizador léxico o scanner) es la parte del compilador que se encarga de reconocer los patrones léxicos (*tokens*) del programa. Junto con el analizador morfológico se entregará un programa escrito en C para probarlo.

Desarrollo de la práctica

1. Codificación de una especificación para Flex.

A continuación se describirá la lista de patrones léxicos (*tokens*) del lenguaje **o**. A partir de ella deberás diseñar las expresiones regulares que los representan. A partir de estas expresiones regulares se codificará en el fichero ***omicron.l*** la especificación correspondiente para la herramienta Flex.

Es importante prestar especial atención al orden en el que se colocan las reglas para la detección de los *tokens* para evitar que unos patrones oculten a otros. A este respecto, se recuerda que Flex para cada token identificado ejecutará la regla que:

- Permita establecer correspondencia con un mayor número de caracteres con la entrada.
- En caso de que varias reglas permitan corresponder exactamente el mismo número de caracteres, se deshace el empate quedándose con aquella que se haya declarado en primer lugar.

Junto a las reglas necesarias para la gestión de los patrones léxicos del lenguaje, se añadirán si es necesario reglas para:

- Ignorar los espacios, tabuladores y saltos de línea.
- Ignorar los comentarios (recuérdese que en Ómicron un comentario comienza con // y acaba al final de la línea).
- Gestionar los errores morfológicos (sólo se considerarán los símbolos no permitidos y los identificadores que excedan de la longitud máxima).

La acción asociada a cada regla será un *return* de un valor numérico que identifique el tipo de patrón identificado. Los valores numéricos asociados a cada tipo de token están definidos en el fichero *tokens.h* facilitado.

2. Lista de tokens de o.

Palabras reservadas para clases	
`class`	Para declarar una clase
`inherits`	Para indicar que una clase hereda de otras
`instance_of`	Para instanciar una clase
`discard`	Para liberar los recursos asociados a una instancia
`itself`	Para referirse al objeto dentro del que se encuentra
`hidden`	Cualificador de acceso
`secret`	Cualificador de acceso
`exposed`	Cualificador de acceso
`unique`	Indicador de que el miembro al que se refiere es de clase en oposición a ser de instancia
Otras palabras reservadas (la mayoría tienen el significado que esperarías en un lenguaje de programación procedural de alto nivel)	
`function`	Para declarar una función
`return`	Fin de función
`main`	Para nombrar el programa principal
`int`	Tipos de dato básicos
`boolean`	
`none`	
`array`	Para declarar arrays
`if`	Para las sentencias de control de flujo de programa condicional
`else`	
`while`	Para bucles
`scanf`	Sentencias de entrada / salida
`printf`	
`false`	Constantes booleanas
`true`	

Operadores y delimitadores de un sólo símbolo	
`; # , = ( ) [ ] { } : + - / * < > & ! .`	
Tokens de más de un símbolo	
`==`	Comparación de igualdad
`!=`	Comparación de distinto
`<=`	Comparación de menor o igual
`>=`	Comparación de mayor o igual
`&&`	And
`||`	Or
`->`	Operador de acceso (flecha)
Otros elementos: Recuerda también las reglas de formación para otros elementos válidos en ómicron	
`Identificadores`	Comienzan por letra mayúscula o minúscula, seguida por cualquier cantidad, incluida 0, de letras mayúsculas o minúsculas y dígitos No pueden tener una longitud superior a 50
`Enteros`	La notación habitual en matemáticas
`Comentarios`	Desde el símbolo // hasta el final de la línea donde esté

3. Codificación del programa de prueba.

Una vez construida la especificación Flex para el compilador y generado el programa *lex.yy.c* (que incluye la función *yylex()*), se desarrollará un programa en C para probar el analizador morfológico. Dicho programa recibirá como argumentos los nombres de dos ficheros. Utilizará el primer fichero como entrada, invocará a la función *yylex()* para realizar el análisis morfológico del mismo, y escribirá los resultados del análisis en el segundo fichero. A continuación se describen los ficheros de entrada y salida, y se muestran algunos ejemplos.

Se recuerda asimismo, de acuerdo con lo especificado en la función anterior, que la función *yylex()* devolverá el valor del *return* de la regla que haya ejecutado para el token identificado. Dicho token estará disponible en la variable *yytext*, y la longitud de *yytext* se guardará en *yyleng*. Cuando se vuelva a invocar a *yylex()* el analizador morfológico continuará procesando la entrada donde terminó con el token previo, y esta misma lógica será aplicable hasta que *yylex()* devuelva 0, lo cual indica que ya no hay más entrada que procesar.

3. Descripción del fichero de entrada.

El fichero de entrada contendrá símbolos y palabras clave como las que se han mencionado anteriormente del lenguaje Ómicron en forma de texto plano. Los programas Ómicronpasados como entrada no tienen por qué ser correctos.

4. Funcionalidad del programa.

Haciendo uso del analizador morfológico construido con Flex, el programa de prueba deberá detectar en el fichero de entrada los siguientes patrones:

- Palabras reservadas.
- Símbolos.
- Identificadores (téngase en cuenta las reglas para la construcción de identificadores especificadas en la descripción de la gramática).
- Constantes (enteras o booleanas).
- Errores (símbolos no permitidos e identificadores que excedan de la longitud máxima).

Cada vez que se detecte un token válido, el programa informará de ello, indicando el tipo de token detectado. Cada vez que se detecte un error, el programa avisará del mismo indicando el tipo de error así como la línea y la columna en la que aparece. Es necesario por tanto llevar la cuenta del número de línea y el número de columna en todo momento, para lo cual se recomienda usar las variables *yytext* e *yyleng*.

5. Descripción del fichero de salida.

El fichero de salida tendrá una línea por cada uno de los patrones léxicos detectados en el fichero de entrada. En cada línea se mostrará:

- El tipo de token detectado. Se utilizará el nombre usado en el fichero *tokens.h*.
- El código numérico del token (definido en el fichero *tokens.h*).
- El lexema analizado (valor de la variable *yytext*).

Por ejemplo, si en el fichero de entrada nos encontramos con la palabra *printf*, en el fichero de salida se escribirá:

```
TOK_PRINTF          284  printf
```

En caso de que aparezca algún error, se informará del mismo mostrando el tipo de error, la fila y la columna del fichero en el que aparece (véanse los ejemplos a continuación para ver el formato a seguir para informar de los errores).

<u>Nota:</u> es muy importante respetar el formato de los ficheros de salida.

6. Ejemplos.

Para probar inicialmente el código desarrollado se facilitarán dos ficheros de entrada (*entrada1.txt* y *entrada2.txt*) con sus salidas correspondientes (*salida1.txt* y *salida2.txt*). Suponiendo que el programa ejecutable se denomina *pruebaMorfo*, la siguiente instrucción:

```
pruebaMorfo entrada1.txt misalida1.txt
```

debe generar un fichero con nombre *misalida1.txt* que sea idéntico al fichero *salida1.txt*. En particular, al hacer:

```
diff -Bb salida1.txt misalida1.txt
```

no debe encontrarse ninguna diferencia entre los dos ficheros.

Entrega de la práctica

Se entregará a través de *Moodle* un único fichero comprimido (*.zip*) que deberá cumplir los siguientes requisitos:

- Deberá contener todos los fuentes (ficheros *.l*, *.h* y *.c*) necesarios para resolver el enunciado propuesto. No es necesario incluir el fichero *lex.yy.c* puesto que puede generarse a partir del *.l*.
- Deberá contener un fichero *Makefile* compatible con la herramienta *make* que para el objetivo ***all*** genere el ejecutable de nombre ***pruebaMorfo***.
- El nombre del fichero *.zip* seguirá el siguiente formato***:***

  morfo_YYYY_X.zip

  donde YYYY será el número del grupo (por ejemplo 1311, 1312, 1362,...) y X será el número del grupo de trabajo que tengáis asignado.

Se recuerda al alumno que **las prácticas son incrementales por lo que una práctica aprobada pudiera conllevar errores importantes en las siguientes fases. Es responsabilidad del alumno subsanar totalmente los errores en cada fase, dando correcto cumplimiento a los requerimientos de los enunciados.**