Análisis Léxico: `flex`
Compiladores

Lourdes Del Carmen González Huesca

9 de abril de 2024
Facultad de Ciencias UNAM

Existen muchas variedades de generadores automáticos de *lexers* que dependen del lenguaje de programación usado. También pueden utilizarse para obtener analizadores léxicos de lenguajes particulares.
El más conocido es `lex` creado en 1975 por M. Lesk y E. Schmidt como un estándar para analizar léxicamente expresiones en Unix. Para este ejemplo, usaremos `flex` (*Fast LEXical analyzer generator*) creado en 1987. La descripción de este comando está disponible a través de `man flex` y se pueden revisar muchas referencias [1].

Especificación

El generador recibe como entrada una especificación a través de un archivo con extensión `.l` con el siguiente formato:

```
definitions
%%
rules
%%
user code
```

Es decir, toda especificación consta de tres partes, divididas por `%%`:

- **definiciones**: incluyen las bibliotecas a usarse en el código del usuario o en las acciones
- **reglas**: expresiones regulares junto con la acción correspondiente a cada una de ellas
- **código**: funciones auxiliares

El lenguaje de especificación para analizadores léxicos usa expresiones regulares (donde `a` y `b` son expresiones regulares):

`ab`	concatenación	`[]`	conjunto de caracteres
`a | b`	disyunción	`[^]`	complemento
$a^\star$	cerradura de Kleene	`\`	escape
`a{k}`	repetición k veces	`"..."`	literal
`a?`	opción	`^`	inicio de línea
a^+	cerradura positiva	`$`	fin de línea
`.`	cualquier caracter excepto `\n`	`a/b`	reconoce a si es seguido de b

[1] Las referencias usadas para este resumen son: `https://www.cs.princeton.edu/~appel/modern/c/software/flex/flex.html#SEC1` y `http://alumni.cs.ucr.edu/~lgao/teaching/flex.html`

Las acciones que acompañan a las expresiones regulares están en código C que será usado por el autómata generado. Las reglas (expresión regular y acción) se aplican en orden de aparición en el archivo y si hay dos o más reglas se aplica la primera. Los caracteres no reconocidos se almacenan en la salida estándar.
Veamos dos ejemplos:

- Lenguaje que reconoce palabras y nombres de archivos (archivo `example.l`)

```
%{
#include <stdio.h>
%}

%%
[a-zA-Z][a-zA-Z0-9]*    printf("WORD ");
[a-zA-Z0-9\/.-]+        printf("FILENAME ");
\"                      printf("QUOTE ");
\{                      printf("OBRACE ");
\}                      printf("EBRACE ");
;                       printf("SEMICOLON ");
\n                      printf("\n");
[ \t]+                  /* ignore whitespace */;
%%
```

- Lenguaje que reconoce programas estilo Pascal [2] (archivo `pascaltoy.l`)

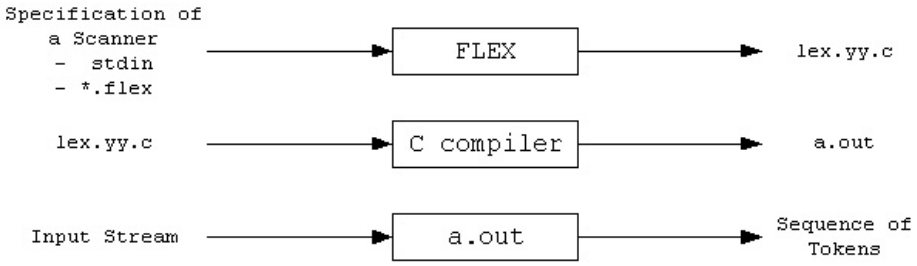
```
/* scanner for a toy Pascal-like language */

%{
/* need this for the call to atof() below */
#include <math.h>
%}

DIGIT    [0-9]
ID       [a-z][a-z0-9]*

%%

{DIGIT}+    {
            printf( "An integer: %s (%d)\n", yytext,
                    atoi( yytext ) );
            }

{DIGIT}+"."{DIGIT}*        {
            printf( "A float: %s (%g)\n", yytext,
                    atof( yytext ) );
            }

if|then|begin|end|procedure|function        {
            printf( "A keyword: %s\n", yytext );
            }
```

[2] Explicado y disponible en `http://web.mit.edu/gnu/doc/html/flex_1.html`

```
{ID}        printf( "An identifier: %s\n", yytext );

"+"|"-"|"*"|"/"   printf( "An operator: %s\n", yytext );

"{"[^}\n]*"}"     /* eat up one-line comments */

[ \t\n]+          /* eat up whitespace */

.           printf( "Unrecognized character: %s\n", yytext );

%%

main( argc, argv )
int argc;
char **argv;
    {
    ++argv, --argc;  /* skip over program name */
    if ( argc > 0 )
            yyin = fopen( argv[0], "r" );
    else
            yyin = stdin;

    yylex();
    }
```

Uso de `flex`

Para generar un lexer mediante `flex` seguiremos la siguiente secuencia de acciones [3]:



En el archivo `lex.yy.c` aparecen varias definiciones, destacamos las siguientes:

- La función `yylex(void)` implementa el analizador léxico haciendo uso de las reglas y acciones definidas.
- Cada que se reconoce una unidad léxica, la cadena reconocida es un valor en `yytext` y la variable `yylen` indica la logitud de la unidad.
- La función `yywrap(void)` es el proceso de decisión:
    - devuelve 1 si el programa se detiene al final del flujo de entrada
    - devuelve 0 si se continua procesando una entrada

[3] Imagen tomada de "FLEX Tutorial" de Lan Gao `http://alumni.cs.ucr.edu/~lgao/teaching/flex.html`

De los ejemplos anteriores:

- Lenguaje que reconoce palabras y nombres de archivos

  ```
  $ flex example.l
  ```

  ```
  $ gcc -w lex.yy.c -o example -lfl
  ```

  Ejemplos de análisis

  `$ echo "foo" | ./example` cuya salida es `WORD`

  `$ echo "{foo.v}" | ./example` cuya salida es `OBRACE FILENAME EBRACE`

- Lenguaje que reconoce programas estilo

  ```
  $ flex pascaltoy.l
  ```

  ```
  $ gcc -w lex.yy.c -o pascal -lfl
  ```

  Ejemplos de análisis

  `$ echo "foo" | ./pascal` cuya salida es `An identifier: foo`

  `$ echo "5/c*(b+4)" | ./pascal` cuya salida es

  ```
  An integer: 5 (5)
  An operator: /
  An identifier: c
  An operator: *
  Unrecognized character: (
  An identifier: b
  An operator: +
  An integer: 4 (4)
  Unrecognized character: )
  ```

Otros generadores léxicos

- `Lex`: generador léxico para Unix que obtiene un lexer en C.
  `https://en.wikipedia.org/wiki/Lex_(software)`
  `http://dinosaur.compilertools.net/lex/index.html` `https://cse.iitkgp.ac.in/~bivasm/notes/LexAndYaccTutorial.pdf`

- `ocamllex`: generador léxico en Ocaml.
  `https://courses.softlab.ntua.gr/compilers/2015a/ocamllex-tutorial.pdf`
  `https://caml.inria.fr/pub/docs/manual-ocaml/lexyacc.html`

- `JLex`: generador para Java escrito en Java.
  `https://www.cs.princeton.edu/~appel/modern/java/JLex/`