El generador de analizadores léxicos *lex*.
IV

Teoría de Autómatas y lenguajes formales
Federico Simmross Wattenberg (fedesim@infor.uva.es)
Universidad de Valladolid

1. Funciones de apoyo

Además de `yylex()`, existen unas pocas funciones que se pueden utilizar en un programa lex, que describimos a continuación. Algunos derivados de lex (notablemente flex) ofrecen varias funciones además de las presentadas, pero no forman parte del estándar de lex.

1.1. yymore()

```
int yymore(void)
```

Cuando se la llama, se le indica a lex que la próxima cadena de entrada que encaje con una regla no debe sobrescribir `yytext`, sino que debe añadirse al final de éste. Es útil en casos en los que una expresión regular se ajusta a una cadena de texto y a una subcadena de ella, por ejemplo, al buscar cadenas de caracteres en C con la expresión regular `\"[^"]*` para la cadena "Hola \" Mundo".

1.2. yyless()

```
int yyless(int n)
```

Vuelve a introducir en la entrada todos los caracteres de `yytext`, excepto los *n* primeros. Dicho de otra manera, `yyless()` devuelve a la entrada los *(yyleng-n)* últimos caracteres de `yytext`. A primera vista, puede parecer que `yyless(0)` hace lo mismo que REJECT, pero `yyless(0)` no impide que se vuelva a utilizar la misma regla continuamente, por lo que puede conducir a un bucle infinito. Sin embargo, el hecho de utilizar `yyless()` no provoca que todo el analizador sea más lento, como ocurre al utilizar REJECT.

1.3. input()

```
int input(void)
```

Devuelve el siguiente carácter de la entrada, y avanza en 1 el puntero de lex. Puede ser útil cuando es posible que se necesite hacer algo con los caracteres que haya después de una expresión regular encontrada. Aunque no es el mejor ejemplo, puede venir bien para saltar los comentarios de un programa en C.

1.4. unput()

```
int unput(int c)
```

Pone en la entrada el carácter *c*, que será el próximo que lea lex.

Nota: Dependiendo de la versión de lex que se utilice, es posible que se destruya el contenido de `yytext` al utilizar `unput()`. De forma notable, esto ocurre en `flex`, que es la versión incluida en los sistemas linux (a no ser que se cambie su comportamiento por defecto mediante la opción adecuada).

1.5. yywrap()

```
int yywrap(void)
```

`yywrap()` es una función que se utiliza para decidir qué debe hacer `yylex()` cuando se alcanza el final del fichero de entrada. Las opciones posibles son:

a) que `yylex()` finalice y devuelva el control a la función llamante (normalmente `main()`). Esto ocurre cuando `yywrap()` devuelve 1.

b) que `yylex()` continúe procesando la entrada de `yyin` como si no hubiera sucedido nada. Esto ocurre cuando `yywrap()` devuelve 0. Para que esto funcione, `yywrap()` debe llevar a cabo las acciones necesarias para que `yyin` apunte a alguna entrada que seguir procesando.

La función `yywrap()` que existe por defecto devuelve siempre 1, por eso `yylex()` finaliza en cuanto acaba de procesar un fichero de entrada. Si queremos construir un analizador que procese varios ficheros, uno detrás de otro, la manera idónea es proporcionar nuestra propia función yywrap() que:

1- Cierre el fichero de entrada actual (`yyin`).
2- Abra el siguiente fichero de entrada (con `fopen()`).
3- Apunte yyin al fichero recién abierto.
4- Devuelva 0.

`yylex()` seguirá procesando la entrada de `yyin` hasta que alcance el fin de fichero e `yywrap()` devuelva 1.

1.6. yylex()

```
int yylex(void)
```

La función `yylex()` es el analizador léxico en sí. Devuelve 0 cuando se ha alcanzado el fin de fichero y el resultado de llamar a `yywrap()` ha sido 1. `yylex()` puede devolver otros valores si se incluyen sentencias `return` como parte de una acción dentro de la sección de reglas.

2. La función `yylex()` como analizador léxico.

Hasta ahora, siempre hemos utilizado la función yylex() como un programa (casi) completo que realiza prácticamente todo el trabajo; sin embargo no es ésta la forma de operar para la que lex está pensado.

Lex genera un código fuente en C que incluye una función pensada para funcionar como uno de los componentes de un compilador: el analizador léxico. Un analizador léxico como tal, se limita a analizar la entrada y devolver al llamante la cadena encontrada (`yytext`), así como un valor que indica de qué tipo es lo que se ha encontrado.

El llamante realiza una función de más alto nivel que el análisis léxico: el análisis sintáctico. Para llevarlo a cabo, realiza repetidamente llamadas al analizador léxico, que le devuelve el siguiente lexema de la entrada, y el tipo de lexema al que pertenece.

La acción habitual en toda regla lex, debe ser entonces, una sentencia `return(tipo)`, para devolver el tipo de lexema encontrado al analizador sintáctico. Con todo esto, una primera idea para construir un compilador de C puede ser la siguiente:

```
%{
#define IF     1
#define ELSE   2
#define NUM    3
#define OPREL  4
#define PARAB  5
#define PARCE  6
%}

digito          [0-9]
num             {digito}+
letra           [A-Za-z]
ident           {letra}({letra|digito))*

%%
if              return IF;
else            return ELSE;
{num}           return NUM;
{ident}         return IDENT;
\(              return PARAB;
\)              return PARCE;

%%
main() {
        int ret;

        while((ret=yylex())!=0) {
                switch(ret) {
                        case IF:
                                .
                                .
                                .
                                break;
                        case ELSE:
                                .
                                .
                                .
                                break;
                        .
                        .
                        .
                }
        }
}
```

3. Ejercicios

1- Construir un programa en C que, apoyándose en un analizador léxico generado por lex, ponga en mayúsculas las palabras reservadas de un fichero fuente en C cualquiera, respetando las siguientes condiciones:

- El fichero de entrada se proporciona como primer argumento al programa, o bien como entrada estándar si el usuario no especifica ningún argumento.

- En la salida no deben aparecer los comentarios del código fuente original.

- El analizador léxico puede encargarse de la tarea de eliminar los comentarios, pero no de pasar a mayúsculas las palabras reservadas.

Palabras reservadas de C estándar ANSI:

```
auto      double  int       struct
break     else    long      switch
case      enum    register  typedef
char      extern  return    union
const     float   short     unsigned
continue  for     signed    void
default   goto    sizeof    volatile
do        if      static    while
```