

# UNIX/Linux & C Programming:

## Chapter n: lex

---

**Coverage:** [UPE] Chapter 8 and [LYT] pp. 1-10, 33-35

**Acknowledgment:** Most of the material in these lecture notes comes from Tom Niemann's [Lex and Yacc Tutorial](#).  
[ePaperPress](#)

---

### Outline

---

- introduction to scanners and parsers
  - examples in `lex`
    - `cat`
      - used built-in macro `ECHO`
      - added file input (`yyin`)
    - line numbering
      - fixed off-by-one error (built in `yylineno` increments as soon as it reads a `\n`)
      - `%option yylineno`
    - `wc`: use of built-in variable `yylen`
    - counting identifiers: added a macro for a RE
    - simple strings and C strings
- 

### UNIX tools for automatically generating scanners and parsers

---

- `lex`: generates a scanner (lexical analyzer or lexer) given a specification of the tokens using REs
  - `yacc` (yet another compiler compiler): generates a parser (syntactic analyzer) given a specification of the grammar
-

---

## Structure of a lex specification:

---

```
/* definitions */

%%

/* a set of pattern-action rules */

%%

/* subroutines */
```

---

## Our first lex program: (cat version 0)

---

```
%%

%%

/* called by lex when EOF reached */
int yywrap (void) {
    /* convention is to return 1 */
    return 1;
}

int main (void) {
    /* main entry point for lex */
    yylex();
    return 0;
}
```

---

## Noop

---

```
/* noop.l */

%%
```

```
. { }
\n { }

%%

int yywrap() {
    return 1;
}

int main() {
    yylex();
    return 0;
}
```

---

## cat (version 1)

---

```
/* cat.l (version 1) */

%%

.    /* match any character except newline */ printf ("%s", yytext);
\n   /* match newline */ printf ("\n");

%%

int yywrap (void) {
    return 1;
}

int main (void) {
    yylex();
    return 0;
}
```

---

## Running lex (to produce the scanner)?

---

```
$ flex cat.l # produces lex.yy.c
$ gcc lex.yy.c # produces a.out, the executable for the scanner
$ ./a.out # runs the scanner
```

or use a Makefile (more on this later)

---

## cat (version 2)

---

```
/* cat2.l (version 2) */

%%

. ECHO;
\n ECHO;

%%

int yywrap (void) {
    return 1;
}

int main (int argc, char** argv) {
    yyin = fopen (argv[1], "r");
    yylex();
    fclose (yyin);
    return 0;
}
```

---

## cat (version 3)

---

```
/* cat3.l (version 3) */

%{
int cc=0;
%}

%%
```

```
. { cc++; ECHO; }

\n { cc++; ECHO; }

%%

int yywrap (void) {
    return 1;
}

int main (int argc, char** argv) {
    yyin = fopen (argv[1], "r");
    yylex();
    fclose (yyin);
    printf ("%d characters\n", cc);
    return 0;
}
```

---

## cat -n (version 4)

---

```
/* cat4.l (version 4) */
/* cat -n */

%{
    int cc = 0;
    int lineno = 0;
}%

%%

^.*\n    { cc += strlen(yytext);
           printf ("%d %s", ++lineno, yytext); }

%%

int yywrap() {
    return 1;
}

int main(int argc, char** argv) {
    yyin = fopen (argv[1], "r");
    yylex();
}
```

```
    printf ("%d characters.\n", cc);
    fclose(yyin);
    return 0;
}
```

---

## cat -n (version 5)

---

```
/* cat5.l (version 5) */
/* cat -n */

%option yylineno

%{
int cc = 0;
}%

%%
^.*\n { cc += strlen(yytext);
        printf("%4d\t%s", yylineno-1, yytext); }

%%

int yywrap (void) {
    return 1;
}

int main (int argc, char** argv) {
    yyin = fopen (argv[1], "r");
    yylex();
    printf ("%d characters.\n", cc);
    fclose (yyin);
    return 0;
}
```

---

## Word count

---

```
%{
int cc = 0;
```

```

int wc = 0;
int lc = 0;
%}

%%

[ \t]    { cc++; }

\n       { lc++; cc++; }

[^ \t\n]+ { wc++; cc += yyleng; /* count anything but whitespace */ }

%%

int yywrap() {
    return 1;
}

int main(int argc, char** argv) {
    yyin = fopen (argv[1], "r");
    yylex();
    printf ("%8d%8d%8d\n", lc, wc, cc);
    fclose(yyin);
    return 0;
}

```

---

## Pattern overlap

---

```

%{
int cc = 0;
int wc = 0;
int lc = 0;
%}

%%

[ ]      { printf("Found a space.\n"); }

[ \t]    { cc++; }

\n       { lc++; cc++; }

```

```
[^ \t\n]+ { wc++; cc += yyleng; /* count anything but whitespace */ }

%%

int yywrap() {
    return 1;
}

int main(int argc, char** argv) {
    yyin = fopen (argv[1], "r");
    yylex();
    printf ("%8d%8d%8d\n", lc, wc, cc);
    fclose(yyin);
    return 0;
}
```

---

## Matching identifiers

---

```
alpha [_a-zA-Z]
alphanumeric [_a-zA-Z0-9]
digit [0-9]

%{
int idcount=0;
}%

%%

{alpha}({alpha}|{digit})* {idcount++; printf ("%s\n", yytext);}
{alpha}{alphanumeric}*    {idcount++; printf ("%s\n", yytext);}

. {}
\n {}

%%

int yywrap (void) {
    return 1;
}
```



```

int main (int argc, char** argv) {
    yyin = fopen (argv[1], "r");
    yylex();
    fclose(yyin);
    printf ("This program contains %d identifiers.\n", idcount);
    return 0;
}

```

---

## Matching quoted strings

---

```

char* yylval;
%}

%%

["][^"\n]*["\n]    { printf (":%s:\n", yytext);
                    yylval = strdup(yytext+1);
                    /* if ( yylval[strlen(yylval)-1] == '"' ) { */
                    if (yylval[yyleng-2] == '"') {
                        /* yylval[strlen(yylval)-1] = '\0'; */
                        yylval[yyleng-2] = '\0';
                        printf (":%s:\n", yylval);
                    } else {
                        warning("Invalid string:");
                        printf (":%s:\n", yylval);
                    }
                }

\n    { }
.     { }

%%

int yywrap() {
    return 1;
}

int warning (char* s) {
    fprintf (stderr, "%s\n", s);
    return 2;
}

```

```

int main(int argc, char** argv) {
    /* flex -d to enable debuggin{ statements */
    yy_flex_debug = 1;
    yylex();
    return 0;
}

%{
#include<string.h>
extern int yy_flex_debug;
char* yylval;
%}

%%

["][^"\n]*["]      { printf (":%s:\n", yytext);
                    yylval = strdup(yytext+1);
                    /* yylval[strlen(yylval)-1] = '\0'; */
                    yylval[yyleng-2] = '\0';
                    printf (":%s:\n", yylval); }

["][^"\n]*[\n]      { printf (":%s:\n", yytext);
                    warning("Invalid string:");
                    printf (":%s:\n", yytext+1); }

\n  { }
.   { }

%%

int yywrap() {
    return 1;
}

int warning (char* s) {
    fprintf (stderr, "%s\n", s);
    return 2;
}

int main(int argc, char** argv) {
    /* flex -d to enable debugging statements */
    yy_flex_debug = 1;
    yylex();

```

```
    return 0;
}
```

---

## States

---

- %s ONE creates the (regular) start state ONE
- `rules that do not have start states can apply in *any* state' ([LY] p. 172)
- %x TWO creates the *exclusive* start state TWO
- `a rule with no start state is not matched when an exclusive state is active' ([LY] p. 172)

```
%{
%}
```

```
%X ONE
%X TWO
```

```
%%
```

```
a {BEGIN ONE; printf("start a\n"); }
```

```
b {BEGIN TWO; printf("start b\n"); }
```

```
<TWO>a { printf ("two a\n"); BEGIN 0; }
<TWO>b { printf ("two b\n"); BEGIN 0; }
<ONE>a { printf ("one a\n"); BEGIN TWO; }
<ONE>b { printf ("one b\n"); BEGIN TWO; }
```

```
%%
```

```
int yywrap() {
    return 1;
}
```

```
int main() {
    yylex();
    return 0;
}
```

---

# Matching C strings

---

```
%{
extern int yy_flex_debug;
char buf[100];
char* s = NULL;

%}

%X STRING

%%

\"                { BEGIN STRING; s = buf; }

<STRING>\\\"      { *s++ = '\\\"'; printf("escaped quote\n"); }
<STRING>\\n      { printf("escaped newline\n"); }
<STRING>\\n      { *s++ = '\\n'; printf("newline\n"); }
<STRING>\\t      { *s++ = '\\t'; printf("tab\n"); }

<STRING>\"        { *s = '\\0';
                  BEGIN 0;
                  printf ("Found '%s'\n", buf); }

<STRING>\\n      { fprintf (stderr, "Invalid string.\n"); }

<STRING>.        { *s++ = *yytext; }

\\n              { }
.                { }

%%

int yywrap() {
    return 1;
}

int main() {
    yy_flex_debug = 0;
    yylex();
    return 0;
}
```

---

## Dependency graph for C strings

---

```
graph TD; all --> a.out; a.out --> lex.yy.c; lex.yy.c --> Cstrings.l
```

---

## Makefile for C strings

---

```
SRC = Cstrings.l
CC = gcc
LEX = flex
LEX_FLAGS = -d
OBJ = lexer

all: $(OBJ)

$(OBJ): lex.yy.c
    $(CC) -o $(OBJ) lex.yy.c

lex.yy.c: $(SRC)
    $(LEX) $(LEX_FLAGS) $(SRC)

clean:
    @-rm lex.yy.c $(OBJ)
```

---

## Pattern matching primitives

---

(ref. [LYT] Table 1, p. 7)

<b>Metacharacter</b>	<b>Matches</b>
.	any character except newline
\n	newline
*	zero or more copies of the preceding expression
+	one or more copies of the preceding expression
?	zero or one copy of the preceding expression
^	beginning of line
\$	end of line
a b	a or b
(ab)+	one or more copies of ab (grouping)
"a+b"	literal "a+b" (C escapes still work)
[ ]	character class

---

## Pattern matching examples

---

(ref. [LYT] Table 2, p. 7)

<b>Expression</b>	<b>Matches</b>
abc	abc
abc*	ab, abc, abcc, abccc, ...
abc+	abc, abcc, abccc, abcccc, ...
a(bc)+	abc, abcbc, abcbcbc, ...
a(bc)?	a, abc
[abc]	one of: a, b, c

[a-z]	any letter, a through z
[a\ -z]	one of: a, -, z
[-az]	one of: -, a, z
[A-Za-z0-9]+	one or more alphanumeric characters
[ \t\n]+	whitespace
[^ab]	anything except: a, b
[a^b]	a, ^, b
[a b]	a,  , b
a b	a, b

---

## lex predefined variables

---

(ref. [LYT] Table 3, p. 10)

Name	Function
int yylex(void)	call to invoke lexer, returns token
char* yytext	pointer to matched string
yylen	length of matched string
yyval	value associated with token
int yywrap(void)	wrapup, return 1 if done, 0 if not done
FILE* yyout	output file
FILE* yyin	input file
INITIAL	initial start condition
BEGIN condition	switch start condition
ECHO	write matched string

## References

[LYT] T. Niemann. [Lex and Yacc Tutorial](#). [ePaperPress](#).

[LY] J.R. Levine, T. Mason, and D. Brown. *Lex and Yacc*. O'Reilly, Cambridge, MA, Second edition, 1995.

[UPE] B.W. Kernighan and R. Pike. *The UNIX Programming Environment*. Prentice Hall, Upper Saddle River, NJ, Second edition, 1984.

