

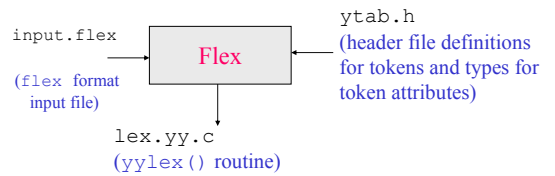
FLEX

Fast Lexical Analyzer Generator

Adapted from material in:
CPU Manual for Flex by Vern Paxson

Overview of Flex

- Scanner generator
- Interface with Parser
 - Scanner called as a subroutine when parser needs the next token.



Flex input file format

- The flex input file consists of three sections, separated by a line with just '%%' in it:

definitions

%%

rules

%%

user code

- Simple Example

%%

```
username printf( "%s", getlogin() );
```

Definitions

- C Code
 - include files
 - global variables
- Regular names defined
- Start Conditions defined (exclusive states, inclusive states)

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

```
DIGIT [0-9]
```

```
ID [a-zA-Z][a-zA-Z0-9_]*
```

```
%x INCOMMENT
```

Rules



- This section contains a list of pairs of the form:

pattern action

where the pattern must be unindented and the action must begin on the same line. The pattern ends at the first non-escaped whitespace character; the remainder of the line is its action.

- A pattern is an extended regular expression; an action is an arbitrary C statement.
 - If the action is empty, then when the pattern is matched, the input token is simply discarded.
 - If an input character matches no pattern, then the scanner writes a copy of the token to the output.

Auxiliary Routines



- The user code section is simply copied to 'lex.yy.c' verbatim. It is used as companion routines which call or are called by the scanner. The presence of this section is optional; if it is missing, the second '%' in the input file may be skipped too.

- Start State**

- Mechanism for *conditionally* activating rules.

- Any rule whose pattern is prefixed with "<sc>" will only be active when the scanner is in the start condition named "sc".

```
<STRING>[^]* { /* eat up the string body ... */ ... }
```

A Simple Example



```
int num_lines = 0, num_chars = 0;

%%
\n ++num_lines; ++num_chars;
. ++num_chars;

%%
main() {
  yylex();
  printf( "# of lines = %d, # of chars = %d\n",
          num_lines, num_chars );
}
```

Generating Scanner



```
UNIX% flex count.flex
UNIX% gcc lex.yy.c -lfl
UNIX% a.out < count.flex
# of lines= 12, # of characters= 250
```

- Using Cygwin tools on PC:

```
W2K% flex count.flex
W2K% gcc lex.yy.c -lfl
W2K% ./a.exe < count.flex
# of lines= 12, # of characters= 250
```

Start State Example



Here is a scanner which recognizes (and discards) C comments while maintaining a count of the current input line.

```
%x comment
%%
int line_num = 1;

"/*"
<comment>[^\n]*
<comment>"*" + [^\n]*
<comment>\n
<comment>"*" + "/"
%%
```

BEGIN(comment);	Rule 1
/* eat anything that's not a '*' */	Rule 2
/* eat up '*'s not followed by '/'s */	Rule 3
++line_num;	Rule 4
BEGIN(INITIAL);	Rule 5

Output of "self-scan"



```
%x comment
%%
int line_num = 1;

"/*"
<comment>[^\n]*
<comment>"*" + [^\n]*
<comment>\n
<comment>"*" + "/"
%%
```

"/*" @BEGIN(comment);
<comment>[^\n]* /* eat anything that's not a '*' */
<comment>"*" + [^\n]* /* eat up '*'s not followed by '/'s */
 <comment>\n ++line_num;
 <comment>"*" + "/" BEGIN(INITIAL);
 %%

"Self-scanning" comment.flex



```
%x comment
%%
int line_num = 1;

"/*"
<comment>[^\n]*
<comment>"*" + [^\n]*
<comment>\n
<comment>"*" + "/"
%%
```

BEGIN(comment);
 /* eat anything that's not a '*' */
 /* eat up '*'s not followed by '/'s */
 ++line_num;
 BEGIN(INITIAL);

printRulesStr.flex



```
%x comment
%%
int line_num = 1;
printf(" INITIAL: Default ");
"/*"
<comment>[^\n]*
<comment>"*" + [^\n]*
<comment>\n
<comment>"*" + "/"
%%
```

{BEGIN(comment);
 printf(" R 1 : |%s|, COMMENT : ", yytext); }
 printf(" R 2 : |%s|", yytext);
 printf(" R 3 : |%s|", yytext);
 printf(" R 4 : |%s| \n, COMMENT : ", yytext);
 {BEGIN(INITIAL);
 printf(" R 5 : |%s|, INITIAL : Default ", yytext);}
 .
 \n
 printf("\n INITIAL : Default ");
 %%