NAME

flex, lex - fast lexical analyzer generator

SYNOPSIS

flex [-bcdfinpstvFILT8 -C[efmF] -Sskeleton] *[filename ...]*

DESCRIPTION

flex is a tool for generating *scanners:* programs which recognized lexical patterns in text. *flex* reads the given input files, or its standard input if no file names are given, for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code, called *rules. flex* generates as output a C source file, **lex.yy.c,** which defines a routine **yylex().** This file is compiled and linked with the **-lfl** library to produce an executable. When the executable is run, it analyzes its input for occurrences of the regular expressions. Whenever it finds one, it executes the corresponding C code.

For full documentation, see **flexdoc(1).** This manual entry is intended for use as a quick reference.

OPTIONS

flex has the following options:

-b Generate backtracking information to *lex.backtrack.* This is a list of scanner states which require backtracking and the input characters on which they do so. By adding rules one can remove backtracking states. If all backtracking states are eliminated and **-f** or **-F** is used, the generated scanner will run faster.

-c is a do-nothing, deprecated option included for POSIX compliance.

NOTE: in previous releases of *flex* **-c** specified table-compression options. This functionality is now given by the **-C** flag. To ease the the impact of this change, when *flex* encounters **-c,** it currently issues a warning message and assumes that **-C** was desired instead. In the future this "promotion" of **-c** to **-C** will go away in the name of full POSIX compliance (unless the POSIX meaning is removed first).

-d makes the generated scanner run in *debug* mode. Whenever a pattern is recognized and the global **yy_flex_debug** is non-zero (which is the default), the scanner will write to *stderr* a line of the form:

```
--accepting rule at line 53 ("the matched text")
```

The line number refers to the location of the rule in the file defining the scanner (i.e., the file that was fed to flex). Messages are also generated when the scanner backtracks, accepts the default rule, reaches the end of its input buffer (or encounters a NUL; the two look the same as far as the scanner's concerned), or reaches an end-of-file.

-f specifies (take your pick) *full table* or *fast scanner.* No table compression is done. The result is large but fast. This option is equivalent to **-Cf** (see below).

-i instructs *flex* to generate a *case-insensitive* scanner. The case of letters given in the *flex* input patterns will be ignored, and tokens in the input will be matched regardless of case. The matched text given in *yytext* will have the preserved case (i.e., it will not be folded).

-n is another do-nothing, deprecated option included only for POSIX compliance.

-p generates a performance report to stderr. The report consists of comments regarding features of the *flex* input file which will cause a loss of performance in the resulting scanner.

-s causes the *default rule* (that unmatched scanner input is echoed to *stdout)* to be suppressed. If the scanner encounters input that does not match any of its rules, it aborts with an error.

-t instructs *flex* to write the scanner it generates to standard output instead of **lex.yy.c.**

-v specifies that *flex* should write to *stderr* a summary of statistics regarding the scanner it generates.

-F specifies that the *fast* scanner table representation should be used. This representation is about as fast as the full table representation (**−f**), and for some sets of patterns will be considerably smaller (and for others, larger). See **flexdoc(1)** for details.

This option is equivalent to **-CF** (see below).

-I instructs *flex* to generate an *interactive* scanner, that is, a scanner which stops immediately rather than looking ahead if it knows that the currently scanned text cannot be part of a longer rule's match. Again, see **flexdoc(1)** for details.

Note, **-I** cannot be used in conjunction with *full* or *fast tables*, i.e., the **-f, -F, -Cf,** or **-CF** flags.

-L instructs *flex* not to generate **#line** directives in **lex.yy.c.** The default is to generate such directives so error messages in the actions will be correctly located with respect to the original *flex* input file, and not to the fairly meaningless line numbers of **lex.yy.c.**

-T makes *flex* run in *trace* mode. It will generate a lot of messages to *stdout* concerning the form of the input and the resultant non-deterministic and deterministic finite automata. This option is mostly for use in maintaining *flex.*

-8 instructs *flex* to generate an 8-bit scanner. On some sites, this is the default. On others, the default is 7-bit characters. To see which is the case, check the verbose **(-v)** output for "equivalence classes created". If the denominator of the number shown is 128, then by default *flex* is generating 7-bit characters. If it is 256, then the default is 8-bit characters.

-C[efmF]
controls the degree of table compression.

-Ce directs *flex* to construct *equivalence classes,* i.e., sets of characters which have identical lexical properties. Equivalence classes usually give dramatic reductions in the final table/object file sizes (typically a factor of 2-5) and are pretty cheap performance-wise (one array look-up per character scanned).

-Cf specifies that the *full* scanner tables should be generated - *flex* should not compress the tables by taking advantages of similar transition functions for different states.

-CF specifies that the alternate fast scanner representation (described in **flexdoc(1))** should be used.

-Cm directs *flex* to construct *meta-equivalence classes,* which are sets of equivalence classes (or characters, if equivalence classes are not being used) that are commonly used together. Meta-equivalence classes are often a big win when using compressed tables, but they have a moderate performance impact (one or two "if" tests and one array look-up per character scanned).

A lone **-C** specifies that the scanner tables should be compressed but neither equivalence classes nor meta-equivalence classes should be used.

The options **-Cf** or **-CF** and **-Cm** do not make sense together - there is no opportunity for meta-equivalence classes if the table is not being compressed. Otherwise the options may be freely mixed.

The default setting is **-Cem,** which specifies that *flex* should generate equivalence classes and meta-equivalence classes. This setting provides the highest degree of table compression. You can trade off faster-executing scanners at the cost of larger tables with the following generally being true:

```
slowest & smallest
      -Cem
      -Cm
      -Ce
      -C
      -C{f,F}e
      -C{f,F}
```

fastest & largest

-C options are not cumulative; whenever the flag is encountered, the previous -C settings are forgotten.

-Sskeleton_file
overrides the default skeleton file from which *flex* constructs its scanners. You'll never need this option unless you are doing *flex* maintenance or development.

SUMMARY OF FLEX REGULAR EXPRESSIONS

The patterns in the input are written using an extended set of regular expressions. These are:

```
x          match the character 'x'
.          any character except newline
[xyz]      a "character class"; in this case, the pattern
             matches either an 'x', a 'y', or a 'z'
[abj-oZ]   a "character class" with a range in it; matches
             an 'a', a 'b', any letter from 'j' through 'o',
             or a 'Z'
[^A-Z]     a "negated character class", i.e., any character
             but those in the class.  In this case, any
             character EXCEPT an uppercase letter.
[^A-Z\n]   any character EXCEPT an uppercase letter or
             a newline
r*         zero or more r's, where r is any regular expression
r+         one or more r's
r?         zero or one r's (that is, "an optional r")
r{2,5}     anywhere from two to five r's
r{2,}      two or more r's
r{4}       exactly 4 r's
{name}     the expansion of the "name" definition
           (see above)
"[xyz]\"foo"
           the literal string: [xyz]"foo
\X         if X is an 'a', 'b', 'f', 'n', 'r', 't', or 'v',
             then the ANSI-C interpretation of \x.
             Otherwise, a literal 'X' (used to escape
             operators such as '*')
\123       the character with octal value 123
\x2a       the character with hexadecimal value 2a
(r)        match an r; parentheses are used to override
             precedence (see below)


rs         the regular expression r followed by the
             regular expression s; called "concatenation"


r|s        either an r or an s


r/s        an r but only if it is followed by an s.  The
             s is not part of the matched text.  This type
             of pattern is called as "trailing context".
^r         an r, but only at the beginning of a line
r$         an r, but only at the end of a line.  Equivalent
```

to "r/\n".

<s>r an r, but only in start condition s (see below for discussion of start conditions)

<s1,s2,s3>r
same, but in any of start conditions s1, s2, or s3

<<EOF>> an end-of-file

<s1,s2><<EOF>>
an end-of-file when in start condition s1 or s2

The regular expressions listed above are grouped according to precedence, from highest precedence at the top to lowest at the bottom. Those grouped together have equal precedence.

Some notes on patterns:

- Negated character classes *match newlines* unless "\n" (or an equivalent escape sequence) is one of the characters explicitly present in the negated character class (e.g., "[^A-Z\n]").
- A rule can have at most one instance of trailing context (the '/' operator or the '$' operator). The start condition, '^', and "<<EOF>>" patterns can only occur at the beginning of a pattern, and, as well as with '/' and '$', cannot be grouped inside parentheses. The following are all illegal:

```
foo/bar$
foo|(bar$)
foo|^bar
<sc1>foo<sc2>bar
```

SUMMARY OF SPECIAL ACTIONS

In addition to arbitrary C code, the following can appear in actions:

- **ECHO** copies yytext to the scanner's output.
- **BEGIN** followed by the name of a start condition places the scanner in the corresponding start condition.
- **REJECT** directs the scanner to proceed on to the "second best" rule which matched the input (or a prefix of the input). **yytext** and **yyleng** are set up appropriately. Note that **REJECT** is a particularly expensive feature in terms scanner performance; if it is used in *any* of the scanner's actions it will slow down *all* of the scanner's matching. Furthermore, **REJECT** cannot be used with the *-f* or *-F* options.

  Note also that unlike the other special actions, **REJECT** is a *branch;* code immediately following it in the action will *not* be executed.
- **yymore()** tells the scanner that the next time it matches a rule, the corresponding token should be *appended* onto the current value of **yytext** rather than replacing it.
- **yyless(n)** returns all but the first *n* characters of the current token back to the input stream, where they will be rescanned when the scanner looks for the next match. **yytext** and **yyleng** are adjusted appropriately (e.g., **yyleng** will now be equal to *n*).
- **unput(c)** puts the character *c* back onto the input stream. It will be the next character scanned.
- **input()** reads the next character from the input stream (this routine is called **yyinput()** if the scanner is compiled using **C++).**

- **yyterminate()** can be used in lieu of a return statement in an action. It terminates the scanner and returns a 0 to the scanner's caller, indicating "all done".

  By default, **yyterminate()** is also called when an end-of-file is encountered. It is a macro and may be redefined.

- **YY_NEW_FILE** is an action available only in <<EOF>> rules. It means "Okay, I've set up a new input file, continue scanning".

- **yy_create_buffer(file, size)** takes a *FILE* pointer and an integer *size*. It returns a YY_BUFFER_STATE handle to a new input buffer large enough to accomodate *size* characters and associated with the given file. When in doubt, use **YY_BUF_SIZE** for the size.

- **yy_switch_to_buffer(new_buffer)** switches the scanner's processing to scan for tokens from the given buffer, which must be a YY_BUFFER_STATE.

- **yy_delete_buffer(buffer)** deletes the given buffer.

VALUES AVAILABLE TO THE USER

- **char *yytext** holds the text of the current token. It may not be modified.

- **int yyleng** holds the length of the current token. It may not be modified.

- **FILE *yyin** is the file which by default *flex* reads from. It may be redefined but doing so only makes sense before scanning begins. Changing it in the middle of scanning will have unexpected results since *flex* buffers its input. Once scanning terminates because an end-of-file has been seen, **void yyrestart(FILE *new_file)** may be called to point *yyin* at the new input file.

- **FILE *yyout** is the file to which **ECHO** actions are done. It can be reassigned by the user.

- **YY_CURRENT_BUFFER** returns a **YY_BUFFER_STATE** handle to the current buffer.

MACROS THE USER CAN REDEFINE

- **YY_DECL** controls how the scanning routine is declared. By default, it is "int yylex()", or, if prototypes are being used, "int yylex(void)". This definition may be changed by redefining the "YY_DECL" macro. Note that if you give arguments to the scanning routine using a K&R-style/non-prototyped function declaration, you must terminate the definition with a semi-colon (;).

- The nature of how the scanner gets its input can be controlled by redefining the **YY_INPUT** macro. YY_INPUT's calling sequence is "YY_INPUT(buf,result,max_size)". Its action is to place up to *max_size* characters in the character array *buf* and return in the integer variable *result* either the number of characters read or the constant YY_NULL (0 on Unix systems) to indicate EOF. The default YY_INPUT reads from the global file-pointer "yyin". A sample redefinition of YY_INPUT (in the definitions section of the input file):

```
%{
#undef YY_INPUT
#define YY_INPUT(buf,result,max_size) \
    { \
    int c = getchar(); \
    result = (c == EOF) ? YY_NULL : (buf[0] = c, 1); \
    }
%}
```

- When the scanner receives an end-of-file indication from YY_INPUT, it then checks the **yywrap()** function. If **yywrap()** returns false (zero), then it is assumed that the function has gone ahead and set up *yyin* to point to another input file, and scanning continues. If it returns true (non-zero), then the scanner terminates, returning 0 to its caller.

  The default **yywrap()** always returns 1. Presently, to redefine it you must first "#undef yywrap", as it is currently implemented as a macro. It is likely that **yywrap()** will soon be defined to be a function rather than a macro.

- YY_USER_ACTION can be redefined to provide an action which is always executed prior to the matched rule's action.
- The macro **YY_USER_INIT** may be redefined to provide an action which is always executed before the first scan.
- In the generated scanner, the actions are all gathered in one large switch statement and separated using **YY_BREAK,** which may be redefined. By default, it is simply a "break", to separate each rule's action from the following rule's.

FILES

flex.skel skeleton scanner.

lex.yy.c generated scanner (called *lexyy.c* on some systems).

lex.backtrack
backtracking information for **-b** flag (called *lex.bck* on some systems).

-lfl library with which to link the scanners.

SEE ALSO

flexdoc(1), lex(1), yacc(1), sed(1), awk(1).

M. E. Lesk and E. Schmidt, *LEX - Lexical Analyzer Generator*

DIAGNOSTICS

reject_used_but_not_detected undefined or

yymore_used_but_not_detected undefined - These errors can occur at compile time. They indicate that the scanner uses **REJECT** or **yymore()** but that *flex* failed to notice the fact, meaning that *flex* scanned the first two sections looking for occurrences of these actions and failed to find any, but somehow you snuck some in (via a #include file, for example). Make an explicit reference to the action in your *flex* input file. (Note that previously *flex* supported a **%used/%unused** mechanism for dealing with this problem; this feature is still supported but now deprecated, and will go away soon unless the author hears from people who can argue compellingly that they need it.)

flex scanner jammed - a scanner compiled with **-s** has encountered an input string which wasn't matched by any of its rules.

flex input buffer overflowed - a scanner rule matched a string long enough to overflow the scanner's internal input buffer (16K bytes - controlled by **YY_BUF_MAX** in "flex.skel").

scanner requires -8 flag - Your scanner specification includes recognizing 8-bit characters and you did not specify the -8 flag (and your site has not installed flex with -8 as the default).

fatal flex scanner internal error--end of buffer missed - This can occur in an scanner which is reentered after a long-jump has jumped out (or over) the scanner's activation frame. Before reentering the scanner, use:

```
yyrestart( yyin );
```

too many %t classes! - You managed to put every single character into its own %t class. *flex* requires that at least one of the classes share characters.

AUTHOR

Vern Paxson, with the help of many ideas and much inspiration from Van Jacobson. Original version by Jef Poskanzer.

See flexdoc(1) for additional credits and the address to send comments to.

DEFICIENCIES / BUGS

Some trailing context patterns cannot be properly matched and generate warning messages ("Dangerous trailing context"). These are patterns where the ending of the first part of the rule matches the beginning of the second part, such as "zx*/xy*", where the 'x*' matches the 'x' at the beginning of the trailing context. (Note that the POSIX draft states that the text matched by such patterns is undefined.)

For some trailing context rules, parts which are actually fixed-length are not recognized as such, leading to the abovementioned performance loss. In particular, parts using '|' or {n} (such as "foo{3}") are always considered variable-length.

Combining trailing context with the special '|' action can result in *fixed* trailing context being turned into the more expensive *variable* trailing context. For example, this happens in the following example:

```
%%
abc     |
xyz/def
```

Use of unput() invalidates yytext and yyleng.

Use of unput() to push back more text than was matched can result in the pushed-back text matching a beginning-of-line ('^') rule even though it didn't come at the beginning of the line (though this is rare!).

Pattern-matching of NUL's is substantially slower than matching other characters.

flex does not generate correct #line directives for code internal to the scanner; thus, bugs in *flex.skel* yield bogus line numbers.

Due to both buffering of input and read-ahead, you cannot intermix calls to <stdio.h> routines, such as, for example, **getchar(),** with *flex* rules and expect it to work. Call **input()** instead.

The total table entries listed by the **-v** flag excludes the number of table entries needed to determine what rule has been matched. The number of entries is equal to the number of DFA states if the scanner does not use **REJECT,** and somewhat greater than the number of states if it does.

REJECT cannot be used with the *-f* or *-F* options.

Some of the macros, such as **yywrap(),** may in the future become functions which live in the **-lfl** library. This will doubtless break a lot of code, but may be required for POSIX-compliance.

The *flex* internal algorithms need documentation.