



Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'lex.1p' command

\$ man lex.1p

LEX(1P) POSIX Programmer's Manual LEX(1P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lex ? generate programs for lexical tasks (DEVELOPMENT)

SYNOPSIS

lex [-t] [-n|-v] [file...]

DESCRIPTION

The lex utility shall generate C programs to be used in lexical processing of character input, and that can be used as an interface to yacc. The C programs shall be generated from lex source code and conform to the ISO C standard, without depending on any undefined, unspecified, or implementation-defined behavior, except in cases where the code is copied directly from the supplied source, or in cases that are documented by the implementation. Usually, the lex utility shall write the program it generates to the file lex.yy.c; the state of this file is unspecified if lex exits with a non-zero exit status. See the EXTENDED DESCRIPTION section for a complete description of the lex input language.

OPTIONS

The lex utility shall conform to the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines, except for Guideline 9.

The following options shall be supported:

- n Suppress the summary of statistics usually written with the -v option. If no table sizes are specified in the lex source code and the -v option is not specified, then -n is implied.
- t Write the resulting program to standard output instead of lex.yy.c.
- v Write a summary of lex statistics to the standard output. (See the discussion of lex table sizes in Definitions in lex.) If the -t option is specified and -n is not specified, this report shall be written to standard error. If table sizes are specified in the lex source code, and if the -n option is not specified, the -v option may be enabled.

OPERANDS

The following operand shall be supported:

- file A pathname of an input file. If more than one such file is specified, all files shall be concatenated to produce a single lex program. If no file operands are specified, or if a file operand is '-', the standard input shall be used.

STDIN

The standard input shall be used if no file operands are specified, or if a file operand is '-'. See INPUT FILES.

INPUT FILES

The input files shall be text files containing lex source code, as described in the EXTENDED DESCRIPTION section.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of lex:

- LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of POSIX.1?2017, Section 8.2, Internationalization Variables for the precedence of internationalization variables

used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE

Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions. If this variable is not set to the POSIX locale, the results are unspecified.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), and the behavior of character classes within regular expressions. If this variable is not set to the POSIX locale, the results are unspecified.

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of **LC_MESSAGES**.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If the **-t** option is specified, the text file of C source code output of **lex** shall be written to standard output.

If the **-t** option is not specified:

- * Implementation-defined informational, error, and warning messages concerning the contents of **lex** source code input shall be written to either the standard output or standard error.
- * If the **-v** option is specified and the **-n** option is not specified, **lex** statistics shall also be written to either the standard output or standard error, in an implementation-defined format. These statistics may also be generated if table sizes are specified with a

'%' operator in the Definitions section, as long as the -n option is not specified.

STDERR

If the -t option is specified, implementation-defined informational, error, and warning messages concerning the contents of lex source code input shall be written to the standard error.

If the -t option is not specified:

1. Implementation-defined informational, error, and warning messages concerning the contents of lex source code input shall be written to either the standard output or standard error.
2. If the -v option is specified and the -n option is not specified, lex statistics shall also be written to either the standard output or standard error, in an implementation-defined format. These statistics may also be generated if table sizes are specified with a '%' operator in the Definitions section, as long as the -n option is not specified.

OUTPUT FILES

A text file containing C source code shall be written to lex.yy.c, or to the standard output if the -t option is present.

EXTENDED DESCRIPTION

Each input file shall contain lex source code, which is a table of regular expressions with corresponding actions in the form of C program fragments.

When lex.yy.c is compiled and linked with the lex library (using the -l operand with c99), the resulting program shall read character input from the standard input and shall partition it into strings that match the given expressions.

When an expression is matched, these actions shall occur:

- * The input string that was matched shall be left in yytext as a null-terminated string; yytext shall either be an external character array or a pointer to a character string. As explained in Definitions in lex, the type can be explicitly selected using the %array or %pointer declarations, but the default is implementation-de

fined.

- * The external int yyleng shall be set to the length of the matching string.
- * The expression's corresponding program fragment, or action, shall be executed.

During pattern matching, lex shall search the set of patterns for the single longest possible match. Among rules that match the same number of characters, the rule given first shall be chosen.

The general format of lex source shall be:

```
Definitions %% Rules %% UserSubroutines
```

The first "%%" is required to mark the beginning of the rules (regular expressions and actions); the second "%%" is required only if user sub? routines follow.

Any line in the Definitions section beginning with a <blank> shall be assumed to be a C program fragment and shall be copied to the external definition area of the lex.yy.c file. Similarly, anything in the Definitions section included between delimiter lines containing only "%{" and "%}" shall also be copied unchanged to the external definition area of the lex.yy.c file.

Any such input (beginning with a <blank> or within "%{" and "%}" delimiter lines) appearing at the beginning of the Rules section before any rules are specified shall be written to lex.yy.c after the declarations of variables for the yylex() function and before the first line of code in yylex(). Thus, user variables local to yylex() can be declared here, as well as application code to execute upon entry to yylex().

The action taken by lex when encountering any input beginning with a <blank> or within "%{" and "%}" delimiter lines appearing in the Rules section but coming after one or more rules is undefined. The presence of such input may result in an erroneous definition of the yylex() function.

C-language code in the input shall not contain C-language trigraphs.

The C-language code within "%{" and "%}" delimiter lines shall not contain any lines consisting only of "%}", or only of "%%".

Definitions in lex

Definitions appear before the first "%%" delimiter. Any line in this section not contained between "%{" and "%}" lines and not beginning with a <blank> shall be assumed to define a lex substitution string.

The format of these lines shall be:

name substitute

If a name does not meet the requirements for identifiers in the ISO C standard, the result is undefined. The string substitute shall replace the string {name} when it is used in a rule. The name string shall be recognized in this context only when the braces are provided and when it does not appear within a bracket expression or within double-quotes.

In the Definitions section, any line beginning with a <percent-sign> ('%') character and followed by an alphanumeric word beginning with either 's' or 'S' shall define a set of start conditions. Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall define a set of exclusive start conditions. When the generated scanner is in a %s state, patterns with no state specified shall be also active; in a %x state, such patterns shall not be active. The rest of the line, after the first word, shall be considered to be one or more <blank>-separated names of start conditions. Start condition names shall be constructed in the same way as definition names. Start conditions can be used to restrict the matching of regular expressions to one or more states as described in Regular Expressions in lex.

Implementations shall accept either of the following two mutually-exclusive declarations in the Definitions section:

%array Declare the type of yytext to be a null-terminated character array.

%pointer Declare the type of yytext to be a pointer to a null-terminated character string.

The default type of yytext is implementation-defined. If an application refers to yytext outside of the scanner source file (that is, via an extern), the application shall include the appropriate %array or %pointer declaration in the scanner source file.

Implementations shall accept declarations in the Definitions section for setting certain internal table sizes. The declarations are shown in the following table.

Table: Table Size Declarations in lex

Declaration	Description	Minimum Value
%p n	Number of positions	2500
%n n	Number of states	500
%a n	Number of transitions	2000
%e n	Number of parse tree nodes	1000
%k n	Number of packed character classes	1000
%o n	Size of the output array	3000

In the table, n represents a positive decimal integer, preceded by one or more <blank> characters. The exact meaning of these table size numbers is implementation-defined. The implementation shall document how these numbers affect the lex utility and how they are related to any output that may be generated by the implementation should limitations be encountered during the execution of lex. It shall be possible to determine from this output which of the table size values needs to be modified to permit lex to successfully generate tables for the input language. The values in the column Minimum Value represent the lowest values conforming implementations shall provide.

Rules in lex

The rules in lex source files are a table in which the left column contains regular expressions and the right column contains actions (C program fragments) to be executed when the expressions are recognized.

```
ERE action
ERE action
...
```

The extended regular expression (ERE) portion of a row shall be separated from action by one or more <blank> characters. A regular expres?

sion containing <blank> characters shall be recognized under one of the following conditions:

- * The entire expression appears within double-quotes.
- * The <blank> characters appear within double-quotes or square brackets, etc.
- * Each <blank> is preceded by a <backslash> character.

User Subroutines in lex

Anything in the user subroutines section shall be copied to lex.yy.c following yylex().

Regular Expressions in lex

The lex utility shall support the set of extended regular expressions (see the Base Definitions volume of POSIX.1?2017, Section 9.4, Extended Regular Expressions), with the following additions and exceptions to the syntax:

"..." Any string enclosed in double-quotes shall represent the characters within the double-quotes as themselves, except that <backslash>-escapes (which appear in the following table) shall be recognized. Any <backslash>-escape sequence shall be terminated by the closing quote. For example, "\01"1" represents a single string: the octal value 1 followed by the character '1'.

<state>r, <state1,state2,...>r

The regular expression r shall be matched only when the program is in one of the start conditions indicated by state, state1, and so on; see Actions in lex. (As an exception to the typographical conventions of the rest of this volume of POSIX.1?2017, in this case <state> does not represent a metavariable, but the literal angle-bracket characters surrounding a symbol.) The start condition shall be recognized as such only at the beginning of a regular expression.

r/x The regular expression r shall be matched only if it is followed by an occurrence of regular expression x (x is the instance of trailing context, further defined below). The token

followed by the character '1'.

The order of precedence given to extended regular expressions for lex differs from that specified in the Base Definitions volume of POSIX.1?2017, Section 9.4, Extended Regular Expressions. The order of precedence for lex shall be as shown in the following table, from high to low.

Note: The escaped characters entry is not meant to imply that these are operators, but they are included in the table to show their relationships to the true operators. The start condition, trailing context, and anchoring notations have been omitted from the table because of the placement restrictions described in this section; they can only appear at the beginning or ending of an ERE.

Table: ERE Precedence in lex

Extended Regular Expression	Precedence
collation-related bracket symbols [= =] [: :] [. .]	
escaped characters	\<special character>
bracket expression	[]
quoting	"..."
grouping	()
definition	{name}
single-character RE duplication	* +
concatenation	
interval expression	{m,n}
alternation	

The ERE anchoring operators '^' and '\$' do not appear in the table. With lex regular expressions, these operators are restricted in their use: the '^' operator can only be used at the beginning of an entire regular expression, and the '\$' operator only at the end. The operators apply to the entire regular expression. Thus, for example, the pattern

"(^abc)|(def\$)" is undefined; it can instead be written as two separate rules, one with the regular expression "^abc" and one with "def\$", which share a common action via the special '|' action (see below). If the pattern were written "^abc|def\$", it would match either "abc" or "def" on a line by itself.

Unlike the general ERE rules, embedded anchoring is not allowed by most historical lex implementations. An example of embedded anchoring would be for patterns such as "(^\\)foo(|\$)" to match "foo" when it exists as a complete word. This functionality can be obtained using existing lex features:

```
^foo/[ \\n] |
" foo"/[ \\n] /* Found foo as a separate word. */
```

Note also that '\$' is a form of trailing context (it is equivalent to "\\n") and as such cannot be used with regular expressions containing another instance of the operator (see the preceding discussion of trailing context).

The additional regular expressions trailing-context operator '/' can be used as an ordinary character if presented within double-quotes, "/"; preceded by a <backslash>, "\v"; or within a bracket expression, "[/]".

The start-condition '<' and '>' operators shall be special only in a start condition at the beginning of a regular expression; elsewhere in the regular expression they shall be treated as ordinary characters.

Actions in lex

The action to be taken when an ERE is matched can be a C program fragment or the special actions described below; the program fragment can contain one or more C statements, and can also include special actions.

The empty C statement ';' shall be a valid action; any string in the lex.yy.c input that matches the pattern portion of such a rule is effectively ignored or skipped. However, the absence of an action shall not be valid, and the action lex takes in such a condition is undefined.

The specification for an action, including C statements and special actions, can extend across several lines if enclosed in braces:

ERE <one or more blanks> { program statement
program statement }

The program statements shall not contain unbalanced curly brace preprocessing tokens.

The default action when a string in the input to a lex.yy.c program is not matched by any expression shall be to copy the string to the output. Because the default behavior of a program generated by lex is to read the input and copy it to the output, a minimal lex source program that has just "%%" shall generate a C program that simply copies the input to the output unchanged.

Four special actions shall be available:

| ECHO; REJECT; BEGIN

| The action '|' means that the action for the next rule is the action for this rule. Unlike the other three actions, '|' cannot be enclosed in braces or be <semicolon>-terminated; the application shall ensure that it is specified alone, with no other actions.

ECHO; Write the contents of the string yytext on the output.

REJECT; Usually only a single expression is matched by a given string in the input. REJECT means ``continue to the next expression that matches the current input'', and shall cause whatever rule was the second choice after the current rule to be executed for the same input. Thus, multiple rules can be matched and executed for one input string or overlapping input strings. For example, given the regular expressions "xyz" and "xy" and the input "xyz", usually only the regular expression "xyz" would match. The next attempted match would start after z. If the last action in the "xyz" rule is REJECT, both this rule and the "xy" rule would be executed. The REJECT action may be implemented in such a fashion that flow of control does not continue after it, as if it were equivalent to a goto to another part of yylex(). The use of REJECT may result in somewhat larger and slower scanners.

BEGIN The action:

BEGIN newstate;

switches the state (start condition) to newstate. If the string newstate has not been declared previously as a start condition in the Definitions section, the results are unspecified. The initial state is indicated by the digit '0' or the token INITIAL.

The functions or macros described below are accessible to user code included in the lex input. It is unspecified whether they appear in the C code output of lex, or are accessible only through the -ll operand to c99 (the lex library).

int yylex(void)

Performs lexical analysis on the input; this is the primary function generated by the lex utility. The function shall return zero when the end of input is reached; otherwise, it shall return non-zero values (tokens) determined by the actions that are selected.

int yymore(void)

When called, indicates that when the next input string is recognized, it is to be appended to the current value of yytext rather than replacing it; the value in yyleng shall be adjusted accordingly.

int yyless(int n)

Retains n initial characters in yytext, NUL-terminated, and treats the remaining characters as if they had not been read; the value in yyleng shall be adjusted accordingly.

int input(void)

Returns the next character from the input, or zero on end-of-file. It shall obtain input from the stream pointer yyin, although possibly via an intermediate buffer. Thus, once scanning has begun, the effect of altering the value of yyin is undefined. The character read shall be removed from the input stream of the scanner without any processing by the scanner.

int unput(int c)

Returns the character 'c' to the input; yytext and yyleng are undefined until the next expression is matched. The result of using unput() for more characters than have been input is unspecified.

The following functions shall appear only in the lex library accessible through the -ll operand; they can therefore be redefined by a conforming application:

int yywrap(void)

Called by yylex() at end-of-file; the default yywrap() shall always return 1. If the application requires yylex() to continue processing with another source of input, then the application can include a function yywrap(), which associates another file with the external variable FILE *yyin and shall return a value of zero.

int main(int argc, char *argv[])

Calls yylex() to perform lexical analysis, then exits. The user code can contain main() to perform application-specific operations, calling yylex() as applicable.

Except for input(), unput(), and main(), all external and static names generated by lex shall begin with the prefix yy or YY.

EXIT STATUS

The following exit values shall be returned:

- 0 Successful completion.
- >0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

Conforming applications are warned that in the Rules section, an ERE without an action is not acceptable, but need not be detected as erroneous by lex. This may result in compilation or runtime errors.

The purpose of input() is to take characters off the input stream and discard them as far as the lexical analysis is concerned. A common use is to discard the body of a comment once the beginning of a comment is

recognized.

The lex utility is not fully internationalized in its treatment of regular expressions in the lex source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer interpret the regular expressions given in the lex source according to the environment specified when the lexical analyzer is executed, but this is not possible with the current lex technology. Furthermore, the very nature of the lexical analyzers produced by lex must be closely tied to the lexical requirements of the input language being described, which is frequently locale-specific anyway. (For example, writing an analyzer that is used for French text is not automatically useful for processing other languages.)

EXAMPLES

The following is an example of a lex program that implements a rudimentary scanner for a Pascal-like syntax:

```
%{
/* Need this for the call to atof() below. */
#include <math.h>
/* Need this for printf(), fopen(), and stdin below. */
#include <stdio.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);
}
{ID} printf("An identifier: %s\n", yytext);
"+"|"-"|"*"|"|" /" printf("An operator: %s\n", yytext);
"{"[^}\n]*}" /* Eat up one-line comments. */
[\t\n]+ /* Eat up white space. */
. printf("Unrecognized character: %s\n", yytext);
%%

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

```

RATIONALE

Even though the `-c` option and references to the C language are retained in this description, `lex` may be generalized to other languages, as was done at one time for EFL, the Extended FORTRAN Language. Since the `lex` input specification is essentially language-independent, versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are known historical implementations that do so.

The current description of `lex` bypasses the issue of dealing with internationalized EREs in the `lex` source code or generated lexical analyzer. If it follows the model used by `awk` (the source code is assumed to be presented in the POSIX locale, but input and output are in the locale specified by the environment variables), then the tables in the lexical analyzer produced by `lex` would interpret EREs specified in the `lex` source in terms of the environment variables specified when `lex` was executed. The desired effect would be to have the lexical analyzer interpret the EREs given in the `lex` source according to the environment

specified when the lexical analyzer is executed, but this is not possible with the current lex technology.

The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard use of escape sequences.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

There is no detailed output format specification. The observed behavior of lex under four different historical implementations was that none of these implementations consistently reported the line numbers for error and warning messages. Furthermore, there was a desire that lex be allowed to output additional diagnostic messages. Leaving message formats unspecified avoids these formatting questions and problems with internationalization.

Although the %x specifier for exclusive start conditions is not historical practice, it is believed to be a minor change to historical implementations and greatly enhances the usability of lex programs since it permits an application to obtain the expected functionality with fewer statements.

The %array and %pointer declarations were added as a compromise between historical systems. The System V-based lex copies the matched text to a yytext array. The flex program, supported in BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements are available for some scanners. Most historical programs should require no change in porting from one system to another because the string being referenced is null-terminated in both cases. (The method used by flex in its case is to null-terminate the token in place by remembering the character that used to come right after the token and replacing it before continuing on to the next scan.) Multi-file programs with external references to yytext outside the scanner source file should continue to operate on their historical systems, but would require one of the new declarations to be considered strictly portable.

The description of EREs avoids unnecessary duplication of ERE details

because their meanings within a lex ERE are the same as that for the ERE in this volume of POSIX.1?2017.

The reason for the undefined condition associated with text beginning with a <blank> or within "%{" and "%}" delimiter lines appearing in the Rules section is historical practice. Both the BSD and System V lex copy the indented (or enclosed) input in the Rules section (except at the beginning) to unreachable areas of the yylex() function (the code is written directly after a break statement). In some cases, the System V lex generates an error message or a syntax error, depending on the form of indented input.

The intention in breaking the list of functions into those that may appear in lex.yy.c versus those that only appear in libl.a is that only those functions in libl.a can be reliably redefined by a conforming application.

The descriptions of standard output and standard error are somewhat complicated because historical lex implementations chose to issue diagnostic messages to standard output (unless -t was given). POSIX.1?2008 allows this behavior, but leaves an opening for the more expected behavior of using standard error for diagnostics. Also, the System V behavior of writing the statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The programmer can always precisely obtain the desired results by using either the -t or -n options.

The OPERANDS section does not mention the use of - as a synonym for standard input; not all historical implementations support such usage for any of the file operands.

A description of the translation table was deleted from early proposals because of its relatively low usage in historical applications.

The change to the definition of the input() function that allows buffering of input presents the opportunity for major performance gains in some applications.

The following examples clarify the differences between lex regular expressions and regular expressions appearing elsewhere in this volume of

POSIX.1?2017. For regular expressions of the form "r/x", the string matching r is always returned; confusion may arise when the beginning of x matches the trailing portion of r. For example, given the regular expression "a*b/cc" and the input "aaabcc", yytext would contain the string "aaab" on this match. But given the regular expression "x*/xy" and the input "xxx", the token xxx, not xx, is returned by some implementations because xxx matches "x*".

In the rule "ab*/bc", the "b*" at the end of r extends r's match into the beginning of the trailing context, so the result is unspecified. If this rule were "ab/bc", however, the rule matches the text "ab" when it is followed by the text "bc". In this latter case, the matching of r cannot extend into the beginning of x, so the result is specified.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, ed, yacc

The Base Definitions volume of POSIX.1?2017, Chapter 5, File Format Notation, Chapter 8, Environment Variables, Chapter 9, Regular Expressions, Section 12.2, Utility Syntax Guidelines

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at <http://www.opengroup.org/unix/online.html>.

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html.

