**Jan napsat makefile**

<https://www.sallyx.org/sally/c/linux/makefile>

<http://jaknaprojekty.davidm.cz/make.html>

Flex česky

<https://atrey.karlin.mff.cuni.cz/~0rfelyus/bifle/flex.html#soubor>

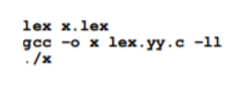
http://martin.lipinsky.cz/skola/fjp/flex.pdf

http://www.ucw.cz/~hubicka/skolicky/skolicka1.html#uvod

Lex manual

<http://dinosaur.compilertools.net/lex/index.html>

<http://malyzelenyhnus.sweb.cz/flex/flex.html>



Makefile

https://cs.wikipedia.org/wiki/Make

https://cs.wikipedia.org/wiki/Make

%%

/\* obsahje reg výraz pravidel

[\ T] + $;¨

Nebo

%%

[\ T] + $;

[\ T] + printf("");

Konečný automat generované pro tento zdroj bude testovat pro obě pravidel najednou, pozorování při ukončení řetězce přířezy či kartách, zda je nebo není znak nové řádky, a provádění požadovanou akci pravidla. První pravidlo splňuje všechny nitky přířezy či karet na konci řádků a druhé pravidlo všechny zbývající struny přířezy či kartách.

Obecný formát zdroje Lex je:

{definice}

%%

{} pravidla

%%

{user} podprogramy

* kde definice a uživatelské podprogramy jsou často vynechány.

V obrysu Lex programů jak je uvedeno výše, pravidla představují rozhodnutí kontrole uživatele; jsou tabulka, ve kterém je levý sloupec obsahuje regulární výrazy (viz bod 3) a pravý sloupec obsahuje akce, fragmenty programu, který má být vykonán, když jsou výrazy uznány. Tak se může objevit individuální pravidlo

integer printf("found keyword INT");

hledat řetězec integer ve vstupním proudu a vytištění zprávy `` nalezeno INT klíčové slovo "" pokaždé, když se objeví. V tomto příkladu je hostitelská procedurální jazyk C a C funkce knihovny printf se používá pro tisk řetězec. Konec projevu je označen první prázdný nebo tabulátoru. V případě, že akce je pouze jeden výraz C, může být jen uvedeny na pravé straně linky; v případě, že je sloučenina, nebo trvá déle než linie, měla by být uzavřen do závorek. Je o něco více užitečné například, že je žádoucí měnit počet slov z Brity americké hláskování. Lex pravidla, jako jsou

colour printf("color");

mechanise printf("mechanize");

petrol printf("gas");

**Lex Regular Expressions.**

The definitions of regular expressions are very similar to those in QED [5]. A regular expression specifies a set of strings to be matched. It contains text characters (which match the corresponding characters in the strings being compared) and operator characters (which specify repetitions, choices, and other features). The letters of the alphabet and the digits are always text characters; thus the regular expression

integer

matches the string integer wherever it appears and the expression

a57D

looks for the string a57D.

Operators. The operator characters are

" \ [ ] ^ - ? . \* + | ( ) $ / { } % < >

and if they are to be used as text characters, an escape should be used. The quotation mark operator (") indicates that whatever is contained between a pair of quotes is to be taken as text characters. Thus

xyz"++"

matches the string xyz++ when it appears. Note that a part of a string may be quoted. It is harmless but unnecessary to quote an ordinary text character; the expression

"xyz++"

is the same as the one above. Thus by quoting every non-alphanumeric character being used as a text character, the user can avoid remembering the list above of current operator characters, and is safe should further extensions to Lex lengthen the list.

An operator character may also be turned into a text character by preceding it with \ as in

xyz\+\+

which is another, less readable, equivalent of the above expressions. Another use of the quoting mechanism is to get a blank into an expression; normally, as explained above, blanks or tabs end a rule. Any blank character not contained within [] (see below) must be quoted. Several normal C escapes with \ are recognized: \n is newline, \t is tab, and \b is backspace. To enter \ itself, use \\. Since newline is illegal in an expression, \n must be used; it is not required to escape tab and backspace. Every character but blank, tab, newline and the list above is always a text character.

Character classes. Classes of characters can be specified using the operator pair []. The construction [abc] matches a single character, which may be a, b, or c. Within square brackets, most operator meanings are ignored. Only three characters are special: these are \ - and ^. The - character indicates ranges. For example,

[a-z0-9<>\_]

indicates the character class containing all the lower case letters, the digits, the angle brackets, and underline. Ranges may be given in either order. Using - between any pair of characters which are not both upper case letters, both lower case letters, or both digits is implementation dependent and will get a warning message. (E.g., [0-z] in ASCII is many more characters than it is in EBCDIC). If it is desired to include the character - in a character class, it should be first or last; thus

[-+0-9]

matches all the digits and the two signs.

In character classes, the ^ operator must appear as the first character after the left bracket; it indicates that the resulting string is to be complemented with respect to the computer character set. Thus

[^abc]

matches all characters except a, b, or c, including all special or control characters; or

[^a-zA-Z]

is any character which is not a letter. The \ character provides the usual escapes within character class brackets.

Arbitrary character. To match almost any character, the operator character . is the class of all characters except newline. Escaping into octal is possible although non-portable:

[\40-\176]

matches all printable characters in the ASCII character set, from octal 40 (blank) to octal 176 (tilde).

Optional expressions. The operator ? indicates an optional element of an expression. Thus

ab?c

matches either ac or abc.

Repeated expressions. Repetitions of classes are indicated by the operators \* and +.

a\*

is any number of consecutive a characters, including zero; while

a+

is one or more instances of a. For example,

[a-z]+

is all strings of lower case letters. And

[A-Za-z][A-Za-z0-9]\*

indicates all alphanumeric strings with a leading alphabetic character. This is a typical expression for recognizing identifiers in computer languages.

Alternation and Grouping. The operator | indicates alternation:

(ab|cd)

matches either ab or cd. Note that parentheses are used for grouping, although they are not necessary on the outside level;

ab|cd

would have sufficed. Parentheses can be used for more complex expressions:

(ab|cd+)?(ef)\*

matches such strings as abefef, efefef, cdef, or cddd; but not abc, abcd, or abcdef.

Context sensitivity. Lex will recognize a small amount of surrounding context. The two simplest operators for this are ^ and $. If the first character of an expression is ^, the expression will only be matched at the beginning of a line (after a newline character, or at the beginning of the input stream). This can never conflict with the other meaning of ^, complementation of character classes, since that only applies within the [] operators. If the very last character is $, the expression will only be matched at the end of a line (when immediately followed by newline). The latter operator is a special case of the / operator character, which indicates trailing context. The expression

ab/cd

matches the string ab, but only if followed by cd. Thus

ab$

is the same as

ab/\n

Left context is handled in Lex by start conditions as explained in section 10. If a rule is only to be executed when the Lex automaton interpreter is in start condition x, the rule should be prefixed by

<x>

using the angle bracket operator characters. If we considered ``being at the beginning of a line'' to be start condition ONE, then the ^ operator would be equivalent to

<ONE>

Start conditions are explained more fully later.

Repetitions and Definitions. The operators {} specify either repetitions (if they enclose numbers) or definition expansion (if they enclose a name). For example

{digit}

looks for a predefined string named digit and inserts it at that point in the expression. The definitions are given in the first part of the Lex input, before the rules. In contrast,

a{1,5}

looks for 1 to 5 occurrences of a.

Finally, initial % is special, being the separator for Lex source segments.

## Lex Actions.

<http://epaperpress.com/lexandyacc/prl.html>

## Příklad souboru Lex [ [upravit překlad](https://en.wikipedia.org/w/index.php?title=Lex_(software)&action=edit&section=3)]

Níže je uveden příklad soubor Lex pro [flex](https://en.wikipedia.org/wiki/Flex_lexical_analyser" \o "Flex lexikální analyzátor) verzi Lex. Rozlišuje řetězce čísel (kladná celá čísla) ve vstupu a jednoduše je vytiskne.

*/ \*\*\* Definice oddílu \*\*\* /*

% {

*/ \* C kód, který má být kopírován verbatim \* /*

#include *<stdio.h>*

% }

*/ \* To říká flex číst pouze jeden vstupní soubor \* /*

% možnost noyywrap

%%

*/ \*\*\* Oddíl Pravidel \*\*\* /*

*/ \* [0-9] + odpovídá řetězce s jedním nebo více číslicemi \* /*

[ 0 - 9 ] + {

*/ \* yytext je řetězec obsahující odpovídající text. \* /*

printf ( " Vidět celé číslo:% s **\ n** " , yytext );

}}

. | \ n { */ \* Ignorujte všechny ostatní znaky. \* /*  }

%%

*/ \*\*\* C Část kódu \*\*\* /*

int main ( void )

{

*/ \* Zavolejte lexeru a pak ukončete. \* /*

yylex ();

**návrat**  0 ;

}}

Je-li tento vstup dána flex, bude převeden do souboru C lex.yy.c. To může být kompilováno do spustitelného souboru, který odpovídá a vygeneruje řetězce celých čísel. Například s ohledem na vstup:

abc123z.! & \* 2gj6

program vytiskne:

Viděl celé číslo: 123

Viděl celé číslo: 2

Viděl celé číslo: 6

Příklad 2