

Using Lex or Flex

Prof. James L. Frankel
Harvard University

Version of 1:07 PM 26-Sep-2016
Copyright © 2016, 2015 James L. Frankel. All rights reserved.

Lex Regular Expressions (1 of 4)

- Special characters are:
 - \ (back slash)
 - " (double quote)
 - . (period)
 - ^ (caret or up arrow)
 - \$ (dollar sign)
 - [(open bracket)
 -] (close bracket)
 - * (asterisk)
 - + (plus sign)
 - ? (question mark)
 - { (open brace)
 - } (close brace)
 - | (vertical bar)
 - / (slash)
 - - (dash or hyphen)
 - ((open parenthesis)
 -) (close parenthesis)

Lex Regular Expressions (2 of 4)

- `c` matches the single non-operator char `c`
- `\c` matches the character `c`
- `"s"` matches the string `s`
- `.` matches any character except newline
- `^` matches beginning of line
- `$` matches end of line
- `[s]` matches any one character in `s`
- `[^s]` matches any one character not in `s`

Lex Regular Expressions (3 of 4)

- r^* matches zero or more strings matching r
- r^+ matches one or more strings matching r
- $r?$ matches zero or one strings matching r
- $r\{m, n\}$ matches between m and n occurrences of r
- r_1r_2 matches r_1 followed by r_2
- $r_1|r_2$ matches either r_1 or r_2
- (r) matches r
- r_1/r_2 matches r_1 when followed by r_2
- $\{name\}$ matches the regex defined by name

Lex Regular Expressions (4 of 4)

- Within square brackets, referred to as a character class, all operators are ignored except for backslash, hyphen (dash), and caret
- Within a character class, backslash will introduce an escape code
- Within a character class, ranges of characters are allowed by using hyphen
 - a-zA-Z
- Within a character class, if caret is the first character in the class, it indicates matching to any character that is not listed in the square brackets
 - In any other position in the class, caret is a normal character in the character class

File Format

- Extension is .lex
- Content consists of three sections, as follows:

<definitions>

%%

<rules>

%%

<user functions>

Definitions Section

- Anything in the <definitions> sections that is delimited by a line with "%{" to a line with "%}" is copied directly to the output C file
 - This allows user functions to be declared here so that they are declared prior to being called from a rule
- Each line in the <definitions> section (other than those between "%{" and "%}") has the format:
<name> <regex>

Rules Section (1 of 2)

- The rules section consists of a sequence of rules
 - Each rule has a regular expression pattern that starts in column one followed by whitespace (space, tab, or newline) and optionally followed by either a C statement or a sequence of C statements enclosed in braces
 - If there is no C statement, then the input is consumed, but no action is taken with that input and the lexer will look for a new token
- When used in a rule, a name enclosed within braces has its associated <regex> substituted
 - This does not happen when the name within braces is quoted
- There is a default rule which matches any character *and copies it to the output*

Rules Section (2 of 2)

- The C code should return the kind of token (referred to as the token type)
- An optional value of the token may be placed in `yylval`
- By default, the type of `yylval` is `int`
 - The type of `yylval` can be changed by using a `#define` with the preprocessor symbol `YYSTYPE`
 - If present, this `#define` should appear at the beginning of the `%{` part of the definitions section

Rules Details

- If two or more regular expression patterns match a string from the input, the rule which matches the longest input string is chosen
- If two or more regular expression patterns match a string from the input and the input strings are of the same length, then the first rule in the <rules> section is chosen
- Remember to include a rule for an action on whitespace

Lex Invocation and Return Value

- Call `yylex()` to invoke the generated lexer
- Lex scans for tokens from `yyin`
 - `yyin` defaults to `stdin`
- Lex continues to scan for tokens until it executes a return statement in a matching rule in the Rules Section or until it reaches end-of-file
 - On end-of-file, flex returns 0
 - Note: this end-of-file behavior is specific to flex

User Functions Section

- Any support functions to be used in the rules section should appear in the User Functions section
- These functions should be declared in the declaration section

Compiling a Lex file

- `lex lexer-standalone.lex` or
`flex lexer-standalone.lex`
- `gcc lex.yy.c -c`
 - `-c` means to create an object file, but do not link
 - object file will have the extension ".o"
- `gcc -pedantic -Wall lex.yy.o lexer.c -lfl -o lexer`
 - `-pedantic` means to issue all warnings demanded by Standard C
 - `-Wall` means to issue many warnings that some users consider questionable
 - `-lfl` means to link with the flex libraries (on some systems, `-ll` may be needed to link with lex libraries)
 - `-o` is used to specify the name of the executable file

Files produced

- lex reads from stdin or from a specified file and produces a lexer named lex.yy.c
- lex.yy.c is source code in the C Programming Language that needs to be compiled
- The user must specify a main program
 - In our example, the main program is in the file named lexer.c
 - This is where yylex is called
 - yylval must be *defined* in this file

Input and Output

- By default, input to lex comes from stdin and output goes to stdout
- The input and output files may be changed
 - FILE *yyin in the input file
 - FILE *yyout is the output file
- An optional function “int yywrap(void)” is called when input is exhausted
 - It should return 1 if lexical analysis is done
 - It should return 0 if more actions are required
 - This allows yyin to be set to a subsequent file and then lex processing to continue with that file

Special Symbols

- `yytext` the matched string as a null terminated string
- `yytext` the length of the matched string
- `yylex()` the name of the generated lexer function
- `yyval` the value of the token matched
- `yyin` the input file
- `yyout` the output file
- `yywrap()` function called on end of input