Lex – Tool for lexical analysis

G Sudha Sadasivam

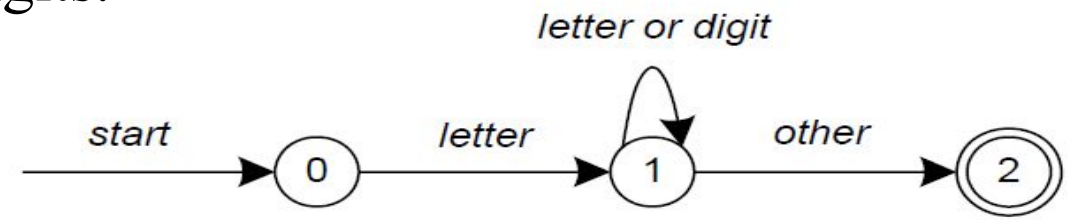
Lex

- lex is a program (generator) that generates lexical analyzers, (widely used on Unix).
- Flex is available for windows also
- It is mostly used with Yacc parser generator.
- Written by Eric Schmidt and Mike Lesk.
- It reads the input stream (specifying the lexical analyzer) and outputs source code implementing the lexical analyzer in the C programming language.
- Lex will read patterns (regular expressions); then produces C code for a lexical analyzer that scans for identifiers.

Lex

- A simple pattern: **letter(letter|digit)***

- Regular expressions are translated by lex to a computer program that mimics an FSA.
- This pattern matches a string of characters that begins with a single letter followed by zero or more letters or digits.



Lex

```
start:   goto state0

state0:  read c
         if c = letter goto state1
         goto state0

state1:  read c
         if c = letter goto state1
         if c = digit goto state1
         goto state2

state2:  accept string
```

- Some limitations, Lex cannot be used to recognize nested structures such as parentheses, since it only has states and transitions between states.
- So, Lex is good at pattern matching, while Yacc is for more challenging tasks.

Lex

Metacharacter	Matches
`.`	any character except newline
`\n`	newline
`*`	zero or more copies of the preceding expression
`+`	one or more copies of the preceding expression
`?`	zero or one copy of the preceding expression
`^`	beginning of line
`$`	end of line
`a|b`	`a` or `b`
`(ab)+`	one or more copies of `ab` (grouping)
`"a+b"`	literal "`a+b`" (C escapes still work)
`[]`	character class

Pattern Matching Primitives

Lex

Expression	Matches
`abc`	`abc`
`abc*`	`ab abc abcc abccc ...`
`abc+`	`abc abcc abccc ...`
`a(bc)+`	`abc abcbc abcbcbc ...`
`a(bc)?`	`a abc`
`[abc]`	one of: `a`, `b`, `c`
`[a-z]`	any letter, a-z
`[a\-z]`	one of: `a`, `-`, `z`
`[-az]`	one of: `-`, `a`, `z`
`[A-Za-z0-9]+`	one or more alphanumeric characters
`[ \t\n]+`	whitespace
`[^ab]`	anything except: `a`, `b`
`[a^b]`	one of: `a`, `^`, `b`
`[a|b]`	one of: `a`, `|`, `b`
`a|b`	one of: `a`, `b`

- Pattern Matching examples.

Lex

……..Definitions section……

%%

……Rules section……..

%%

…….C code section (subroutines)……..

The input structure to Lex.

Lex

Name	Function
`int yylex(void)`	call to invoke lexer, returns token
`char *yytext`	pointer to matched string
`yyleng`	length of matched string
`yylval`	value associated with token
`int yywrap(void)`	wrapup, return 1 if done, 0 if not done
`FILE *yyout`	output file
`FILE *yyin`	input file
`INITIAL`	initial start condition
`BEGIN`	condition switch start condition
`ECHO`	write matched string

Lex predefined variables.

%option noyywrap

Match Id and print its count

```
digit    [0-9]
letter   [A-Za-z]
%{
    int count;
%}
%%
    /* match identifier */
{letter}({letter}|{digit})*        count++;
%%
int main(void) {
    yylex();
    printf("number of identifiers = %d\n", count);
    return 0;
}
```

```
int yywrap(void) {
   return 1;
}
```

- Whitespace must separate the defining term and the associated expression.
- Code in the definitions section is simply copied as-is to the top of the generated C file and must be bracketed with "%{" and "%}" markers.
- substitutions in the rules section are surrounded by braces ({letter}) to distinguish them from literals.

Count char, word, line

```
%{
int nchar, nword, nline;
%}
%%
\n                          { nline++; nchar++; }
[^ \t\n]+                   { nword++, nchar += yyleng; }
.                           { nchar++; }
%%
int main(void) {
yylex();
printf("%d\t%d\t%d\n", nchar, nword, nline);
return 0;
}
int yywrap(void) {
   return 1;
}
```

Print line nos

```
%{
int yylineno;
%}

%%
^(.*)\n          printf("%4d\t%s", ++yylineno, yytext);
%%

int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
fclose(yyin);
}
int yywrap(void) {
   return 1;
}
```

Hex number

```
digit       [0-9]
alpha       [a-fA-F]
hextail         ({digit}|{alpha}){1,8}
hex         0[xX]{hextail}

%%
{hex}  printf("Found a HEX number %s !", yytext);
.   printf("");
%%

main()
{
  printf("Give me your input:\n");
  yylex();  }
```

Flex in windows

- Install code blocks Click here to download codeblocks-16.01mingw-setup
- Install flex Click here to download flex-2.5.4a-1
- Set up systems var path
  - To ……\CodeBlocks\MinGW\bin

  C:\Program Files (x86)\CodeBlocks\MinGW\bin
  - .....\GnuWin32\bin

  C:\Program Files (x86)\GnuWin32\bin
- Create a folder, Open notepad type in a flex program
- - Type in command :- **flex filename.l**
- - Type in command :- **gcc lex.yy.c**

Linking lex&yacc