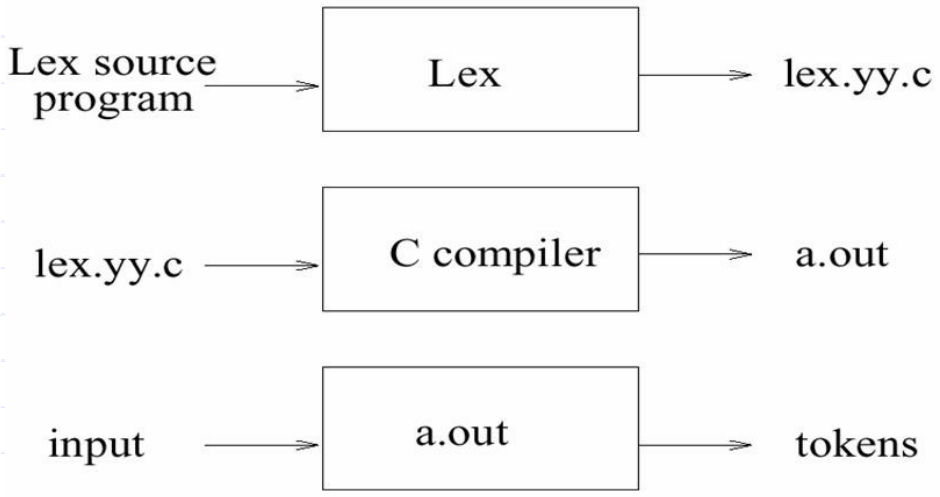
編譯器設計

lex
A Lexical Analyzer Generator

lex : A Tool for Creating Lexical Analyzers

- Lexical analyzers tokenize input streams.
- Regular expressions define tokens.
- Tokens are the terminals of a language.

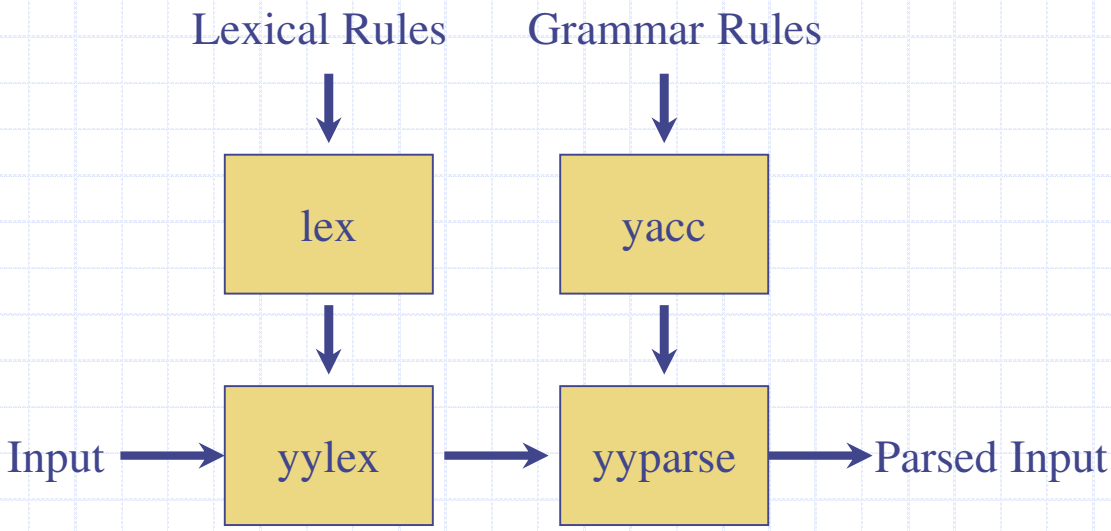
Overview



lex Internals

- Converts regular expressions into NFAs.
- NFAs are implemented as table driven state machines.

lex and yacc



General Format of lex Source

```
%{
     C Declarations and includes
%}

<name> <regexp>
<name> <regexp>
...

%%

<regexp> <action>
<regexp> <action>
...

%%

User Subroutines (C code)
```

General Format of lex Source

- Input specification file is in 3 parts
  - Declarations: Definitions
  - Transition Rules: Token Descriptions and actions
  - Auxiliary Procedures: User-Written code
- Three parts are separated by %%
- Tips: In the first part we define patterns, in the third part we define actions, in the second part we put them together.

General Format of lex Source

- The first and second part must exist, but may be empty, the third part and the second %% are optional.
- A minimum lex program:

```
%%
```

  - It only copies the input to the output unchanged.
- Another trivial example:

```
%%
[ \t]+$;
```

  - It deletes from the input all blanks or tabs at the ends of lines.

A lex Source File Example

```
%{
/*
* Example lex source file
* This first section contains necessary
* C declarations and includes
* to use throughout the lex specifications.
*/
#include <stdio.h>
%}
bin_digit [01]
```


A lex Source File Example

```
%%
{bin_digit}* {
/* match all strings of 0's and 1's */
/* Print out message with matching text
*/
printf("BINARY: %s\n", yytext);
}
([ab]*aa[ab]*bb[ab]*)|([ab]*bb[ab]*aa[ab]*) {
/* match all strings over
* (a,b) containing aa and bb
*/
printf("AABB\n");
}
\n ; /* ignore newlines */
```

A lex Source File Example

```
%%
/*
* Now this is where you want your main program
*/
int main(int argc, char *argv[]) {
/*
 * call yylex to use the generated lexer
 */
yylex();
/*
 * make sure everything was printed
 */
fflush(yyout);
exit(0);
}
```


Running lex

- To run lex on a source file, use the command:
  lex source.l
- This produces the file lex.yy.c which is the C source for the lexical analyzer.
- To compile this, use:
  cc -o scanner -O lex.yy.c -ll

Different Versions Of lex

- AT&T -- lex
  http://www.combo.org/lex_yacc_page/lex.html
- GNU -- flex
  http://www.gnu.org/manual/flex-2.5.4/flex.html
- Find a Win32 version of flex :
  http://www.cygwin.com/

lex.yy.c : What it produces

```
# define YYTYPE unsigned char
struct yywork { YYTYPE verify, advance; } yycrank[] = {
0,0,    0,0,    1,3,    0,0,
0,0,    0,0,    0,0,    0,0,
...

struct yysvf yysvec[] = {
0,      0,      0,
yycrank+-1,     0,              yyvstop+1,
yycrank+-3,     yysvec+1,       yyvstop+3,
yycrank+0,      0,              yyvstop+5,
...

unsigned char yymatch[] = {
00  ,01  ,01  ,01  ,01  ,01  ,01  ,01  ,
01  ,01  ,012 ,01  ,01  ,01  ,01  ,01  ,
...
```

Token Definitions

- Elementary Operations
  - single characters
    - except " \ . $ ^ [ ] - ? * + | ( ) / { } % < >
  - concatenation (put characters together)
  - alternation (a|b|c)
    - [ab] == a|b
    - [a-k] == a|b|c|...|i|j|k
    - [a-z0-9] == any letter or digit
    - [^a] == any character but a

Token Definitions

- Elementary Operations (cont.)
  - NOTE: . matches any character except the newline
  - * -- Kleene Closure
  - + -- Positive Closure
- Examples:
  - [0-9]+"."[0-9]+
    - note: without the quotes it could be any character
  - [\t]+ -- is whitespace
    - (except CR).
    - Yes there is a space inside the box before the \t

Token Definitions

- Special Characters:
  - . -- matches any single character (except newline)
  - " and \ -- quote the part as text
  - \t -- tab
  - \n -- newline
  - \b -- backspace
  - \" -- double quote
  - \\ -- \
  - ? -- this means the preceding was optional
    - ab? == a|ab
    - (ab)? == ab|ϵ

Token Definitions

- Special Characters (cont.)
  - ^ means at the beginning of the line (unless it is inside of a [ ])
  - $ means at the end of the line, same as /\n
  - [^] means anything except
    - \"[^\"]*\" is a double quoted string
  - {n,m} means m through n occurrences
    - a{1,3} is a or aa or aaa
  - {definition} means translation from definition
  - / matches only if followed by right part of /
    - 0/1 means the 0 of 01 but not 02 or 03 or ...
  - ( ) grouping

Definitions

- NAME REG_EXPR
  - digs [0-9]+
  - integer {digs}
  - plain_real {digs}"."{digs}
  - expreal {digs}"."{digs}[Ee][+-]?{digs}
  - real {plainreal}|{expreal}

Definitions

- The definitions can also contain variables and other declarations used by the code generated by lex.
  - These usually go at the start of this section, marked by %{ at the beginning and %} at the end or the line which begins with a blank or tab .
  - Includes usually go here.
  - It is usually convenient to maintain a line counter so that error messages can be keyed to the lines in which the errors are found.

```
%{
    int linecount = 1;
%}
```

Transition Rules

- The code copied into the generated lex program are the same as the definitions section
- The unmatched token is using a default action that ECHO from the input to the output
- A null statement ; will ignore the input
- An action character | indicates that the action for this rule is the action for the next rule

Tokens and Actions

- Example:

```
{real}      return FLOAT;
begin       return BEGIN;
{newline}   linecount++;
{integer}   {
            printf("I found an integer\n");
            return INTEGER;
            }
```

Tokens and Actions

- identifiers used by lex and yacc begin with yy
  - yytext -- a string containing the lexeme
  - yyleng -- the length of the lexeme
  - yylval -- holds the lexical value of the token.
- Example:
  - {integer} {
    printf("I found an integer\n");
    sscanf(yytext, "%d", &yylval);
    return INTEGER;
    }
  - C++ Comments -- // .....
    //.* ;

Lex library function calls

- yylex()
  - default main() contains a return yylex();
- yywarp()
  - called by lexical analyzer if end of the input file
- yyless(n)
  - n characters in yytext are retained
- yymore()
  - the next input expression recognized is to be tacked on to the end of this input

Lex I/O Functions

- c = input()
  - reads another character
- unput(c)
  - puts a character back to be read again a moment later
- output(c)
  - writes a character on an output device

States

- lex allows the user to explicitly declare multiple states

  %s COMMENT
- Default states is INITIAL or 0
- Actions for a matched string may be different states
- BEGIN is used to change state

User Written Code

- The actions associated with any given token are normally specified using statements in C. But occasionally the actions are complicated enough that it is better to describe them with a function call, and define the function elsewhere.
- Definitions of this sort go in the last section of the lex input.

Ambiguous Source Rules

- If 2 rules match the same pattern, lex will use the first rule.
- lex always chooses the longest matching substring for its tokens.
- To override the choice, use action REJECT

```
ex:     she   {s++; REJECT;}
        he    {h++; REJECT;}
        . | \n ;
```

More Example 1

```
int lengs[100];
%%
[a-z]+ lengs[yyleng]++;
. |
\n ;
%%
yywrap()
{
   int i;
   printf("Length No. words\n");
   for(i=0; i<100; i++)
       if (lengs[i] > 0)
            printf("%5d%10d\n",i,lengs[i]);
   return(1);
}
```


More Example 2

```
%{
int charCount=0, wordCount=0, lineCount=0;
%}
word [^ \t\n]+
%%
{word}  {wordCount++; charCount += yyleng; }
[\n]    {charCount++; lineCount++; }
.       {charCount++; }
%%
main() {
        yylex();
        printf("Characters:  %d Words:  %d Lines:%d\n",
           charCount,wordCount,lineCount); }
```

Using yacc with lex

- yacc will call yylex() to get the token from the input so that each lex rule should end with:
  return(token);
  where the appropriate token value is returned.
- An easy way is placing the line:
  #include “lex.yy.c”
  in the last section of yacc input.

Special Notes

- lex on different machines is not created equal.
- Manual page has more advanced topics for the specified lex version.
- Try things early. If you get stuck, ask!

Reference Books

- lex & yacc ,2/e by John R.Levine, Tony Mason & Doug Brown, O'Reilly
- Mastering Regular Expressions, by Jeffrey E.F. Friedl, O'Reilly