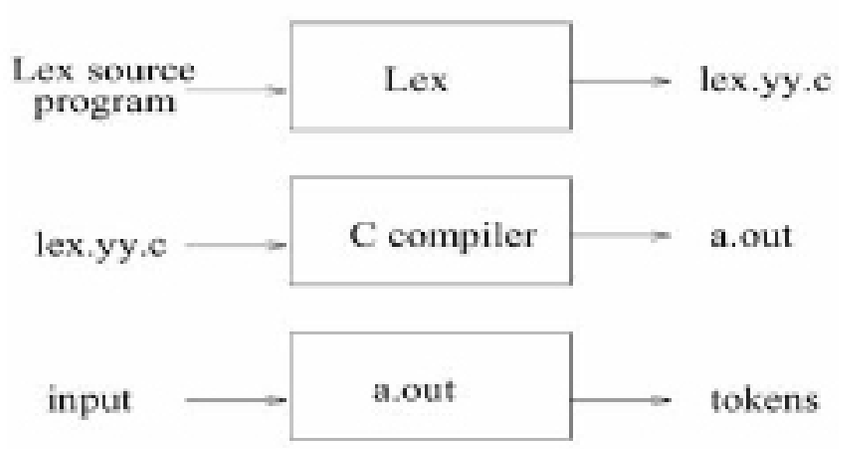
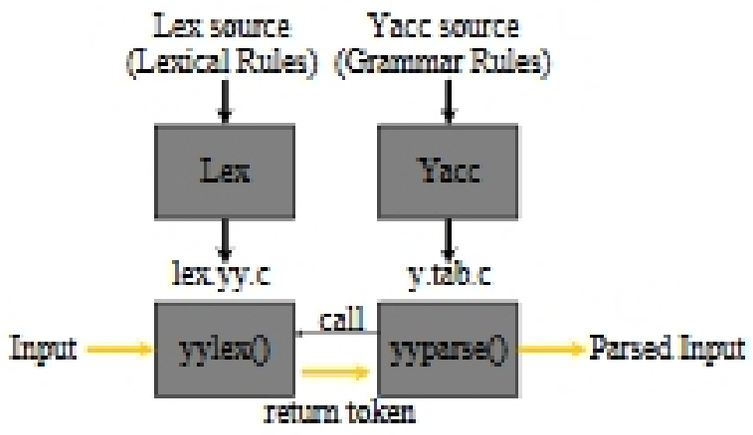
Lex Overview

- Lex is a tool for creating lexical analyzers.
- Lexical analyzers *tokenize* input streams.
- Tokens are the *terminals* of a language.
- Regular expressions define *tokens* .

Usage Paradigm of Lex



To Use Lex and Yacc Together



Lex Internals Mechanism

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

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  ,
...
```

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 prog -O lex.yy.c -ll

Versions and Reference Books

- AT&T lex, GNU flex, and Win32 version
- lex & yacc ,2/e by John R.Levine, Tony Mason & Doug Brown, O'Reilly
- Mastering Regular Expressions, by Jeffrey E.F. Friedl, O'Reilly

General Format of Lex Source

```
%{
    C Declarations and includes
%}

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

%%

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

%%

User Subroutines (C code)
```

- Input specification file is in 3 parts
  - Declarations: Definitions
  - Rules: Token Descriptions and actions
  - Auxiliary Procedures: User-Written code
- Three parts are separated by %%
- Tips: ***The first part defines patterns, the third part defines actions, the second part puts together to express "If we see some pattern, then we do some action".***

Regular Policy of None-translated Source

- Remember that Lex is turning the rules into a program. Any source not intercepted by Lex is ***copied*** into the generated program.
  - Any line which is not part of a Lex rule or action which begins with a blank or tab
  - Anything included between lines containing only %{ and %}
  - Anything after the second %% delimiter

Position of Copied Source

- source input prior to the first %%
  - external to any function in the generated code
- after the first %% and prior to the second %%
  - appropriate place for declarations in the function generated by Lex which contains the actions
- after the second %%
  - after the Lex generated output

Regular Policy of Translated Source

- Various variables or tables whose name prefixed by yy
  - yyleng, yysvec[], yywork[]
- Various functions whose name prefixed by yy
  - yyless(), yymore(), yywarp(), yylex()…
- Various definition whose name are capital
  - BEGIN, INITIAL…

Default Rules and Actions

- The first and second part must exist, but may be empty, the third part and the second %% are optional.
- If the third part dose not contain a main(), -ll will link a default main() which calls yylex() then exits.
- Unmatched patterns will perform a default action, which consists of copying the input to the output

Default Input and Output

- If you don't write your own main() to deal with the input and the output of yylex(), the default input of default main() is **stdin** and the default output of default main() is **stdout**.
  - stdin usually is to be keyboard input
    stdout usually is to be screen output
  - cs20: %./a.out < inputfile > outputfile

Some Simple Lex Source Examples

- A minimum lex program:

  %%

  It only copies the input to the output unchanged.
- A trivial program to deletes three spacing characters:

  %%

  [\t\n];
- Another trivial example:

  %%

  [\t]+$;

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

A General Lex Source 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]
%%
```

```
{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 */
```

```
%%
/*
* 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);
}
```