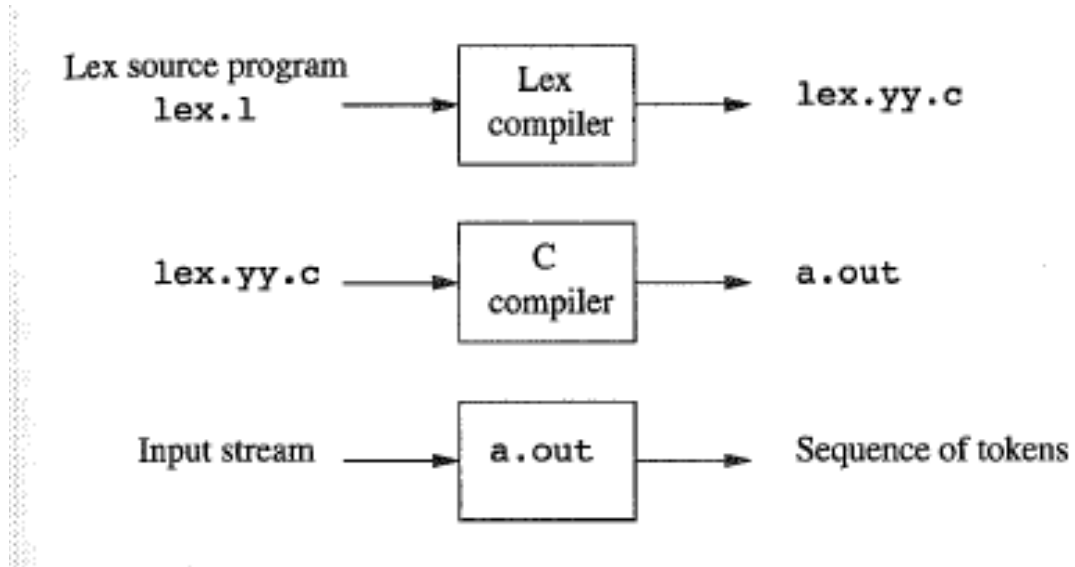


lex: A program that takes lexical definitions as input and generates a scanner (token recognizer).



- Scanner generation
 - lex lex.l
 - you can also use flex, a new version of lex
 - a C file “lex.yy.c” is generated
- Compile
 - cc lex.yy.c -o executable -ll
 - the output is the scanner

A lex input file has the following format

```
% { program related definitions % }
```

RE related definitions

```
%%
```

Translation rules

- RE patterns {actions in C}
- yytext: a global variable points to the text string
- can be passed directly to parser

```
%%
```

Auxiliary functions (can call these functions in actions)

```

%{
    /* definitions of manifest constants
    LT, LE, EQ, NE, GT, GE,
    IF, THEN, ELSE, ID, NUMBER, RELOP */
}%

/* regular definitions */
delim    [ \t\n]
ws       {delim}+
letter   [A-Za-z]
digit    [0-9]
id       {letter}({letter}|{digit})*
number   {digit}+(\.{digit}+)?(E[+-]?{digit}+)?

%%

{ws}     { /* no action and no return */}
if       {return(IF);}
then     {return(THEN);}
else     {return(ELSE);}
{id}     {yylval = (int) installID(); return(ID);}
{number} {yylval = (int) installNum(); return(NUMBER);}
"<"     {yylval = LT; return(RELOP);}
"<="    {yylval = LE; return(RELOP);}
"="      {yylval = EQ; return(RELOP);}
"<>"    {yylval = NE; return(RELOP);}
">"     {yylval = GT; return(RELOP);}
">="    {yylval = GE; return(RELOP);}

%%

int installID() { /* function to install the lexeme, whose
                  first character is pointed to by yytext,
                  and whose length is yyleng, into the
                  symbol table and return a pointer
                  thereto */
}

int installNum() { /* similar to installID, but puts numer-
                   ical constants into a separate table */
}

```