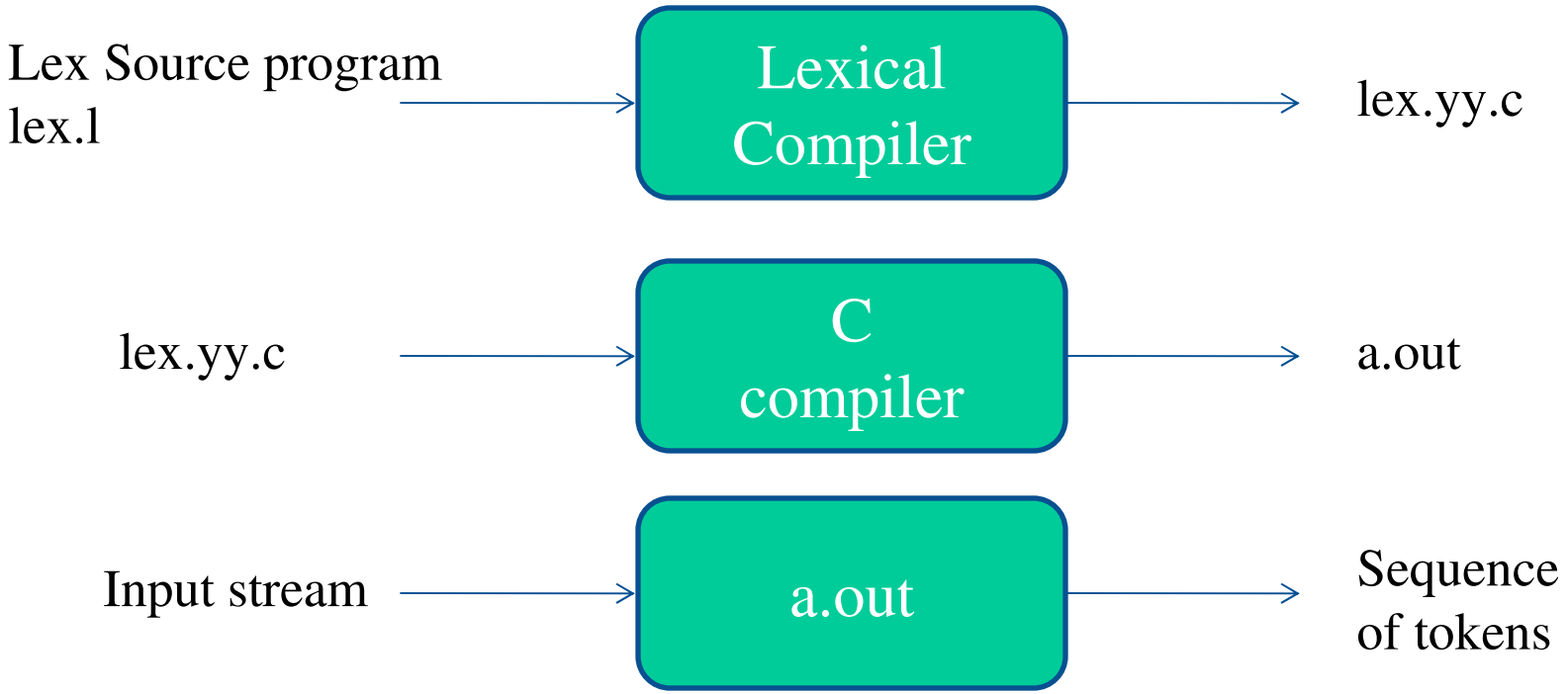
Lexical Analyzer Generator

The Lex

- *Lex* and its newer cousin *flex* are scanner generators
- Systematically translate regular definitions into C source code for efficient scanning
- Generated code is easy to integrate in C applications

Lexical Analyzer Generator - Lex



Lex Specification

- A *lex specification* consists of three parts:

  regular definitions, C declarations in `%{ %}`

  `%%`

  translation rules

  `%%`

  user-defined auxiliary procedures

- The *translation rules* are of the form:

  p_1 { $action_1$ }

  p_2 { $action_2$ }

  …

  p_n { $action_n$ }

Regular Expressions in Lex

x	match the character **x**
\.	match the character **.**
"*string*"	match contents of string of characters
.	match any character except newline
^	match beginning of a line
$	match the end of a line
[xyz]	match one character **x**, **y**, or **z** (use **** to escape **–**)
[^xyz]	match any character except **x**, **y**, and **z**
[a–z]	match one of **a** to **z**
r*****	closure (match zero or more occurrences)
r**+**	positive closure (match one or more occurrences)
r**?**	optional (match zero or one occurrence)
$r_1 r_2$	match r_1 then r_2 (concatenation)
r_1 **|** r_2	match r_1 or r_2 (union)
(r **)**	grouping
r_1 **** r_2	match r_1 when followed by r_2
{d**}**	match the regular expression defined by d

Example Lex Specification 1

Translation rules

Contains the matching lexeme

```
%{
#include <stdio.h>
%}
%%
[0-9]+   { printf("%s\n", yytext); }
.|\n     { }
%%
main()
{ yylex();
}
```

Invokes the lexical analyzer

```
lex spec.l
gcc lex.yy.c -ll
./a.out spec.l
```

Example Lex Specification 2

Translation rules

Regular definition

```
%{
#include <stdio.h>
int ch = 0, wd = 0, nl = 0;
%}
delim      [ \t]+
%%
\n         { ch++; wd++; nl++; }
^{delim}   { ch+=yyleng; }
{delim}    { ch+=yyleng; wd++; }
.          { ch++; }
%%
main()
{ yylex();
  printf("%8d%8d%8d\n", nl, wd, ch);
}
```

Example Lex Specification 3

Translation rules

Regular definitions

```
%{
#include <stdio.h>
%}
digit       [0-9]
letter      [A-Za-z]
id          {letter}({letter}|{digit})*
%%
{digit}+    { printf("number: %s\n", yytext); }
{id}        { printf("ident: %s\n", yytext); }
.           { printf("other: %s\n", yytext); }
%%
main()
{ yylex();
}
```

Example Lex Specification 4

```
%{ /* definitions of manifest constants */
#define LT (256)
…
%}
delim      [ \t\n]
ws         {delim}+
letter     [A-Za-z]
digit      [0-9]
id         {letter}({letter}|{digit})*
number     {digit}+(\.{digit}+)?(E[+\-]?{digit}+)?
%%
{ws}       { }
if         {return IF;}
then       {return THEN;}
else       {return ELSE;}
{id}       {yylval = install_id(); return ID;}
{number}   {yylval = install_num(); 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 install_id()
…
```

Return token to parser

Token attribute

Install **`yytext`** as identifier in symbol table