CS335: A Brief Introduction to Lex and Flex

Swarnendu Biswas

Semester 2019-2020-II

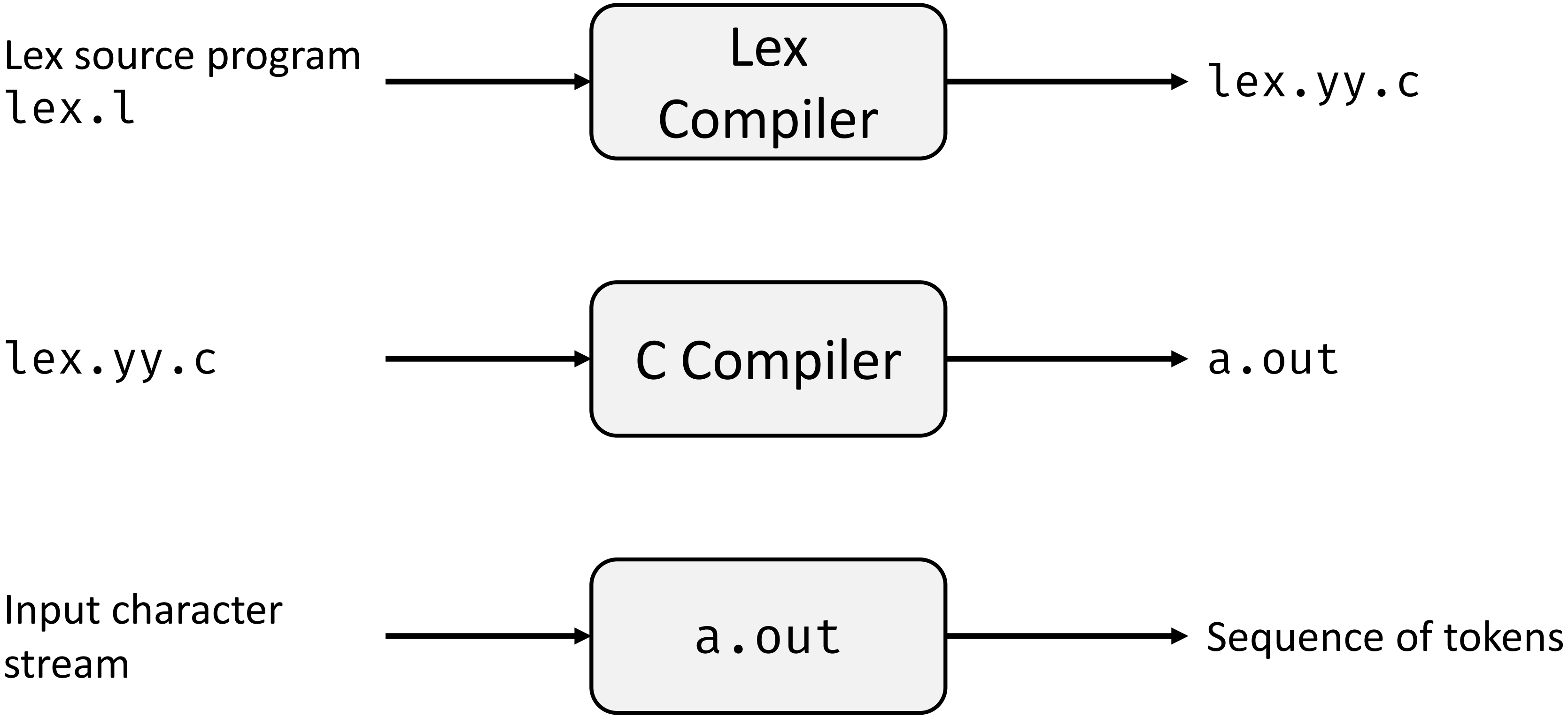
CSE, IIT Kanpur

Content influenced by many excellent references, see References slide for acknowledgements.

Lex and Flex

- Lex and Flex generate programs whose control flow is directed by instances of regular expressions in the input stream
  - Basically, Lex and Flex are lexical analyzer generators
  - Lex and Flex are good at matching patterns
- Lex was originally written by Mike Lesk and Eric Schmidt in 1975
- Flex is an open-source alternative to Lex
  - Lex was originally proprietary software
- Lex and Flex are available on many Unix-like platforms
  - Commonly used with Yacc and Bison, which are parser generators

Block Diagram for Lex



Structure of Lex programs

- Lex program structure

```
definitions
%%
translation rules
%%
user functions
```

required

- Declarations
  - Declaration of variables, manifest constants, and regular definitions

Structure of Lex programs

- Lex program structure

```
definitions
%%
translation rules
%%
user functions
```

required

- Translation rules

```
Pattern { Action }
```

- Each pattern is a regular expression
  - Starts from the first column
- Actions are code fragments
  - Must begin on the same line
  - Multiple sentences are enclosed within braces ({})
- Unmatched input characters are copied to stdout

Structure of Lex programs

- Lex program structure

```
definitions
%%
translation rules
%%
user functions
```

required

- User functions are additional functions used in Actions

A Sample Specification

$$
\begin{aligned}
stmt \ &\longrightarrow \mathbf{if}\ expr\ \mathbf{then}\ stmt \\
&\quad \mid \mathbf{if}\ expr\ \mathbf{then}\ stmt\ \mathbf{else}\ stmt \\
&\quad \mid \epsilon \\
expr &\longrightarrow term\ \mathbf{relop}\ term \\
&\quad \mid term \\
term &\longrightarrow \mathbf{id} \\
&\quad \mid \mathbf{number}
\end{aligned}
$$

$$
\begin{aligned}
&digit \longrightarrow [0{-}9] \\
&digits \longrightarrow digit^{+} \\
&number \longrightarrow digits\ (.\,digits)?\ (E[+-]?\ digits)? \\
&letter \longrightarrow [A - Za - z] \\
&id \longrightarrow letter\ (letter \mid digit)^{*} \\
&if \ \longrightarrow \text{if} \\
&then \longrightarrow \text{then} \\
&else \longrightarrow \text{else} \\
&relop \longrightarrow < \mid > \mid <= \mid >= \mid = \mid <> \\
&ws \ \longrightarrow \ (\text{blank} \mid \text{tab} \mid \text{newline})^{+}
\end{aligned}
$$

Tokens, Lexemes, and Attributes

Lexemes	Token Name	Attribute Value
Any ws	--	--
if	**if**	--
$then$	**then**	--
$else$	**else**	--
Any id	**id**	Pointer to symbol table entry
Any $number$	**number**	Pointer to symbol table entry
$<$	**relop**	LT
<=	**relop**	LE
$=$	**relop**	EQ
$<>$	**relop**	NE
$>$	**relop**	GT
>=	**relop**	GE

Lex Program for Recognizing the Grammar

All definitions within braces is copied to file `lex.yy.c`

```
%{
/∗ 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}+)?
```

Lex Program for Recognizing the Grammar

```
%%
{ws}        {/*no action and no return*/}
if          {printf("%s\n",yytext);}
then        {printf("%s\n",yytext);}
else        {printf("%s\n",yytext);}
{id}        {printf("%s\n",yytext);}
{number}    {printf("%s\n",yytext);}
"<"         {printf("%s\n",yytext);}
"<="        {printf("%s\n",yytext);}
"="         {printf("%s\n",yytext);}
"<>"        {printf("%s\n",yytext);}
">"         {printf("%s\n",yytext);}
">="        {printf("%s\n",yytext);}
%%
```

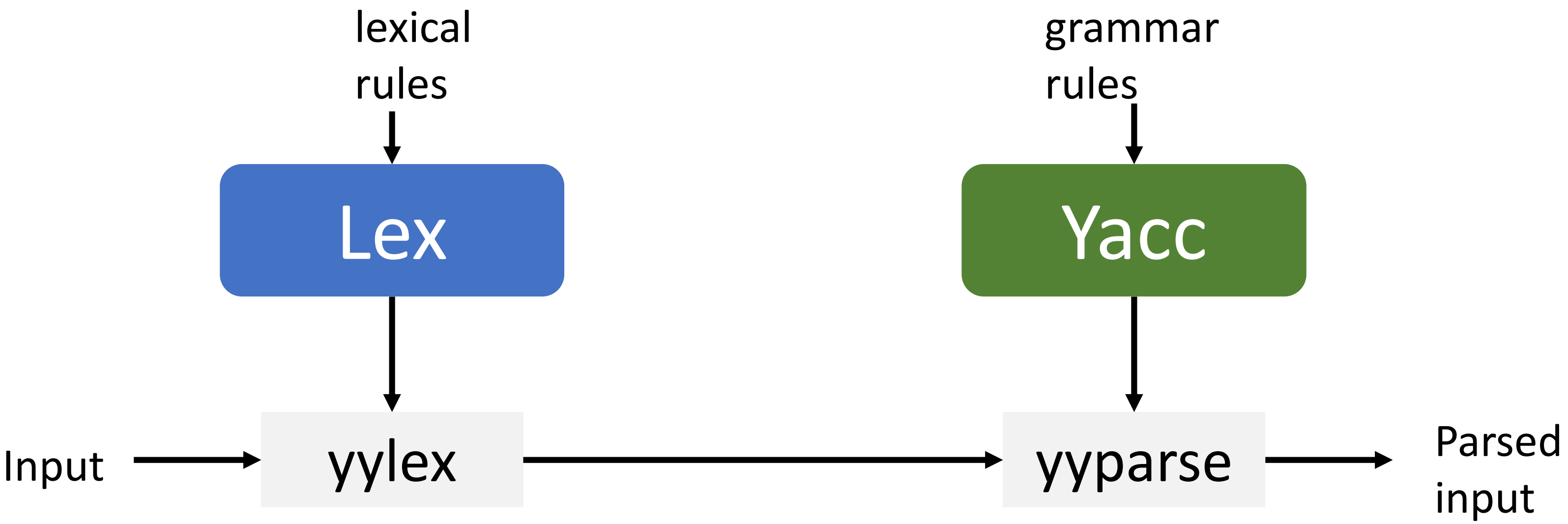
Sample Execution

```
%%
{ws}          {/∗no action and no return∗/}
if            {printf("%s\n",yytext);}
then          {printf("%s\n",yytext);}
else          {printf("%s\n",yytext);}
{id}          {printf("%s\n",yytext);}
{number}      {printf("%s\n",yytext);}
"<"           {printf("%s\n",yytext);}
"<="          {printf("%s\n",yytext);}
"="           {printf("%s\n",yytext);}
"<>"          {printf("%s\n",yytext);}
">"           {printf("%s\n",yytext);}
">="          {printf("%s\n",yytext);}
%%
```

```
❖lex predicate.l; gcc lex.yy.c
❖/a.out
if (a) { x=y+z; } else {x=y+z;}
if
(id: a
){id: x
=
id: y
+id: z
;}else
{id: x
=
id: y
+id: z
;}
```

Lex Workflow

- Lex is invoked
  - Reads remaining input, one character at a time
  - Finds the longest input prefix that matches one of the patterns P_i
    - Executes associated action A_i
    - A_i returns control to the parser, along with the token name
    - Additional information is passed through the global variable `yylval`



Pattern Matching Primitives

RE Syntax	Match
.	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
$	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

Predefined Names in Lex

Name	Function
`int yylex(void)`	Call to invoke lexer, carries out action when match is found, returns token
`char *yytext`	Pointer to the `NULL`-terminated matched string
`int yyleng`	Length of the matched string
`yylval`	Value associated with the token
`int yywrap(void)`	Function which is called when input is exhausted, returns 1 if done, 0 if not done
`FILE *yyout`	Refers to the output file and defaults to stdout
`FILE *yyin`	Input file
`INITIAL`	Initial start condition
`BEGIN`	Condition switch start condition
`ECHO`	Write matched string

Conflict Resolution

- Several prefixes of the input match one or more patterns
  1. Prefer longest match
     - For e.g., prefer “<=“ as a lexeme rather than “<“
  2. If the longest possible prefix matches two or more patterns, prefer the pattern listed first
     - For e.g., make keywords reserved by listing keywords before **id**

Context Sensitivity

- Lex recognizes a small amount of surrounding context
    - For e.g., operators like ^ and $\$$
- Expression ab/cd matches string ab but only if followed by cd
    - Thus ab\$ is same is as $ab/\backslash\mathrm{n}$

START Condition

- “start conditions” can be used to specify that a pattern match only in specific situations
  - Used to activate rules conditionally
  - Any rule prefixed with <S> will be activated only when the scanner is in start condition S
- Define start conditions: $\%Start\ name1, name2, \ldots$
- Recognize rule only when Lex is in start condition $name1$: $<name1>expression$
- Enter a start condition: $BEGIN\ name1$
- Return to normal state: $BEGIN\ 0;$

Use of START Conditions

```
int flag;
%%
^a      {flag = 'a'; ECHO;}
^b      {flag = 'b'; ECHO;}
^c      {flag = 'c'; ECHO;}
\n      {flag =  0 ; ECHO;}
magic   {
          switch (flag) {
            case ‘a’: {
              printf("first"); break; }
            case ‘b’: {
              printf("second"); break; }
            case ‘c’: {
              printf("third"); break; }
            default: ECHO; break;
          }
        }
```

```
%START AA BB CC
%%
^a                {ECHO; BEGIN AA;}
^b                {ECHO; BEGIN BB;}
^c                {ECHO; BEGIN CC;}
\n                {ECHO; BEGIN 0;}
<AA>magic         printf("first");
<BB>magic         printf("second");
<CC>magic         printf("third");
```

Lex vs Flex

Lex	**Flex**
• In Lex, you can provide your own input code and modify the character stream; Flex won't let you do that.	• Rewrite of the Lex tool, but does not reuse code • Supposed to be more efficient • Faster compilation and execution time, smaller transition table

Potential Issues in Using Lex/Flex

- These tools are mostly not reentrant, that is, their states can get corrupted if invoked concurrently by multiple threads

- Generated code may use Unix-specific features
  - You need to disable those features to generate portable code

References

- A. Aho et al. Compilers: Principles, Techniques, and Tools, 2nd edition, Chapter 3.
- http://dinosaur.compilertools.net/lex/index.html
- S. Debray. A brief [f]lex tutorial. https://www2.cs.arizona.edu/~debray/Teaching/CSc453/DOCS/