ceng444 `flex` recitation

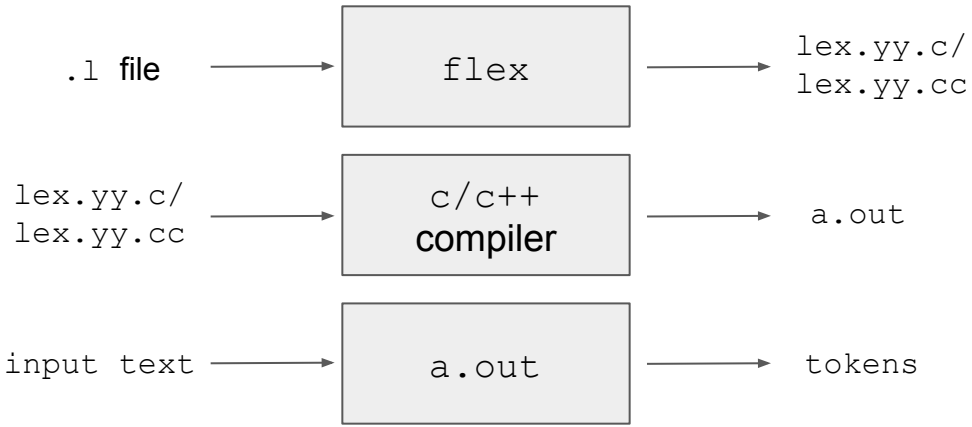
2023/24 spring

outline

- what is `flex`?
- structure of a `.l` file
- `flex` with `c++`
- working with Eclipse IDE & `flex`
- how to have `flex` on your system?
- recommended readings

what is `flex`?

- a tool for generating scanners for lexical analysis
- works with c and c++
- open source (yay!)



a basic pipeline using `flex`

what is `flex`?

bigger picture of the generated scanner

- input is tracked to find a valid token
- tokens are extracted
- global symbol table is updated

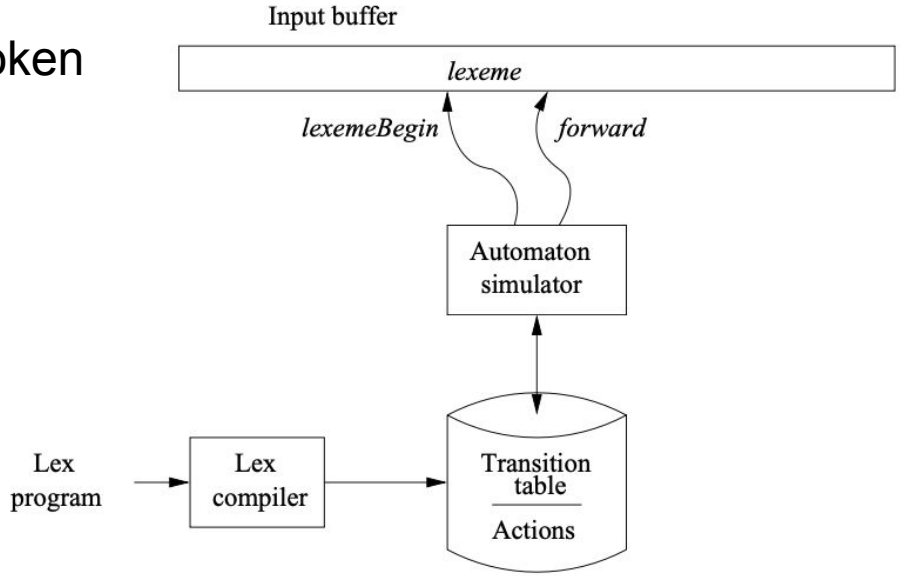


Figure 3.49: A Lex program is turned into a transition table and actions, which are used by a finite-automaton simulator

(from Dragon Book)

structure of a `.l` file

definitions

- regular expressions
- `flex` options
- code to be directly copied between “%{ %}”

rules

- scanned tokens mapped to actions

user code

- yylex() routine
- function definitions that can be used in rules
- can be left blank and carried to a different file

```
<definitions>
%%
<rules>
%%
<user code>
```

`.l` file

code to be directly copied between special parentheses

```
%{
    /* 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 */
}
```

definitions

rules

user code

user defined functions

(from Dragon Book)

Figure 3.23: Lex program for the tokens of Fig. 3.12

```
%{
    /* 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}+)?

%%
```

regular expressions that will be used to describe tokens

(from Dragon Book)

```
%%

{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);}

%%
```

(from Dragon Book)

```
%%

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 */
}
```

(from Dragon Book)

`flex` with `c++`

official example

compiling and running shared example in recitation files from terminal:

- `flex -+ sample01.l`
- `g++ -o lexer lex.yy.cc sample01.cpp`
- `./lexer`

working with Eclipse IDE & `flex`

First, run `flex` from the terminal to generate `lex.yy.cc`

- Create new C/C++ project > C++ Managed Build
- Empty project with Cross GCC
- Import `lex.yy.cc`, `sample01.cpp`, and `MyFlexLexer.h`
- You can also add `.l` and `.txt` files to see them from the IDE, they will be excluded from the default build (if not, you can exclude them manually)
- You can use the integrated terminal to regenerate `lex.yy.cc`, but make sure that you are in the correct folder.

how to have `flex` on your system?

- on a linux machine:
  `<your favourite package manager> install flex`
- on macos:
  using the `brew` package manager works, but clang/gcc difference may cause problems when compiling
- on windows:
  works when integrated in an IDE, tutorials are available online
  → WSL may work, but using a virtual machine is a better idea

department labs have `flex` installed, you can always use them.

recommended readings

- `flex` manual → best resource you can have!
  - `man flex` in your terminal
  - pdf version in the recitation documents
  - online from someone's own build
- chapter 3.5 from the *Dragon Book*
  - `(+)` easy-to-follow tutorial/explanation
  - `(-)` on `c` instead of `c++`

thanks!