Compiler Construction

Lex Tool

Mr. Usman Wajid

usman.wajid@nu.edu.pk

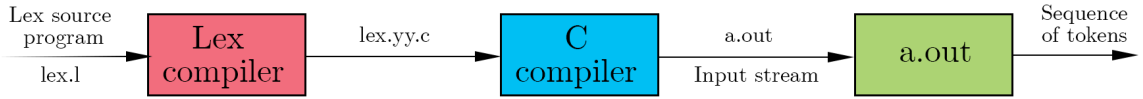
National University
of Computer & Emerging Sciences

lex Tool: Lexical Analyzer Generator

Lex

It is a language to construct a lexical analyzer by using regular expression(s) to recognize tokens.

- The input notations for the `Lex` tool is called Lex language
- Whereas, the tool itself is the `Lex` compiler
- The `Lex` compiler transforms the input patterns into a transition diagram and generates code, in a file called `lex.yy.c`
- `lex.yy.c` simulates the transition diagram



Structure of Lex Programs

- The Structure of a Lex program has three parts,
  1. Declarations
  2. Regular expressions
  3. Subroutines
- each part is separated by double percentage (%%) symbol

```
Declarations
%%
Regular expressions {actions}
%%
Subroutines
```

Structure of Lex Programs

1. **Declarations:**

   - It contains all the C declarations and include
   - Followed by regular definitions, such as, letter $[a - zA - Z]$ and digit $[0 - 9]$

```
%{
int count; // same as normal C program
%}

letter [a-zA-Z]
digit [0-9]

%%
```

Structure of Lex Programs

2. **Regular expressions:**
   - A number of regular expressions with corresponding actions, such as to assign tokens
   - example,

```
%%

{letter}({letter}|{digit})*     printf("%s\tIDENTIFIER\n", yytext);
{digit}+                        printf("%s\tINTEGER\n", yytext);

%%
```

Structure of Lex Programs

3. **Subroutines:**

- An additional tasks can be added to Lex Program, such as,
- taking input from user during runtime or from another source file
- example,

```
%%

int main(int argc, char * argv[]){
        yyin=fopen(argv[1], "r");
        printf("No of identifiers: %d\n", count);
        yylex();
        fclose(yyin);
        return 0
}
```

Installing and Configuring `Lex`

- Also known as `Flex` in the latest versions
- Download, the `Flex` from the following link, https://gnuwin32.sourceforge.net/packages/flex.htm
- Install and add its ”bin” directory path to the ”Windows Environment Variables”
- use the following command to verify successful installation and configuration,
  ```
  flex --version
  ```

Lex:Pattern-matching Primitives

Meta-character	Matches
.	any character except new line
\n	new line
*	zero or more copies of preceding expression
+	one or more copies of preceding expression
?	zero or one copy of the preceding expression
x{m,n}	*x* repeated *m* to *n* times
∧	begging of line or not in a character class
$	end of line
a | b	a or b
(ab)+	one or more copies of ab (grouping)
ab/*cd*	match *ab* but when followed by *cd*
{varname}	substitute a predefined sub-pattern

Lex: Pattern-matching Examples

Expression	Matches
abc	*abc*
*abc**	ab, abc , abcc, abccc …
abc+	abc, abcc, abccc , abcccc …
a(bc)+	abc, abcbc, abcbcbc …
a(bc)?	a, abc
[*abc*]	a, b, c
[*a* − *z*]	a, b, …, z
[*a*\ − *z*]	a, -, z
[*a* − *zA* − *Z*0 − 9]	one or more alphanumeric characters
[\ *t*\ *n*] +	white space
[∧*ab*]	anything except: a, b
[*a* ∧ *b*]	a, ∧, b
[*a* | *b*]	a, |, b
a∧b	a, b