

Appendix B.1

Lex

- Input specification file is in 3 parts
 - ◆ Definitions
 - ◆ Token Descriptions and actions
 - ◆ User-Written code
- Parts are separated by %%
- In the first part we define patterns, in the third part we define actions, in the second part we put them together.

1. Token Definitions

■ Elementary Operations

- ◆ single characters
 - ◆ except . \$ ^ [] - ? * + | () / { } < >
- ◆ concatenation (put characters together)
- ◆ alternation (a|b|c)
 - ◆ [ab] == a|b
 - ◆ [a-k] == a|b|c|...|i|j|k
 - ◆ [a-z0-9] == any letter or digit

■ Elementary Operations (cont.)

- ◆ NOTE: . matches any character except the newline
- ◆ * -- Kleene Closure
- ◆ + -- Positive Closure
- Examples:
 - ◆ [0-9]+ "." [0-9]+
 - ◆ note: without the quotes it could be any character
 - ◆ [\t]+ -- is whitespace
 - ◆ (except CR).
 - ◆ Yes there is a space inside the box before the \t

■ Special Characters:

- ◆ . -- matches any single character (except newline)
- ◆ \t -- tab
- ◆ \n -- newline
- ◆ \" -- double quote
- ◆ \\ -- \
- ◆ ? -- this means the preceding was optional
 - ◆ ab? == a|ab
 - ◆ (ab)? == ab|ε

■ Special Characters (cont.)

- ◆ ^ -- means at the beginning of the line (unless it is inside of a [])
- ◆ \$ means at the end of the line
- ◆ [^] -- means anything except
 - ◆ \"[^\"]*\" is a double quoted string
- Lex always chooses the longest matching substring for its tokens.

2. Definitions

■ NAME REG_EXPR

- ◆ digs [0-9]+
- ◆ integer {digs}
- ◆ plain_real {digs} "." {digs}
- ◆ expreal {digs} "." {digs} [Ee] [+-]? {digs}
- ◆ real {plainreal} | {expreal}

- The definitions can also contain variables and other declarations used by the Code generated by Lex.

- ◆ These usually go at the start of this section, marked by %{ at the beginning and %} at the end.
- ◆ Includes usually go here.
- ◆ It is usually convenient to maintain a line counter so that error messages can be keyed to the lines in which the errors are found.

```
%{  
  int linecount = 1;  
}%
```

3. Tokens and Actions

■ Example:

- ◆ {real} return FLOAT;
- ◆ begin return BEGIN;
- ◆ {newline} linecount++;
- ◆ {integer} {
 - ◆ printf("I found an integer\n");
 - ◆ return INTEGER;}

- identifiers used by Lex and Yacc begin with yy

- ◆ yytext -- a string containing the lexeme
- ◆ yyleng -- the length of the lexeme
- ◆ yylval -- holds the lexical value of the token.

■ Example:

- ◆ {integer} {
 - ◆ printf("I found an integer\n");
 - ◆ sscanf(yytext, "%d", &yylval);
 - ◆ return INTEGER;}

- ◆ C++ Comments -- //

```
◆ //.* ;
```

4. User Written Code

- The actions associated with any given token are normally specified using statements in C. But occasionally the actions are complicated enough that it is better to describe them with a function call, and define the function elsewhere.
- Definitions of this sort go in the last section of the Lex input.

5. A Sample Lex Specification

- **Note:** If 2 rules match the same pattern, Lex will use the first rule.