Chapter 3

The Scanner

A scanner (lexical analyzer) is a program which recognizes patterns in text. Scanners may be hand written or may be automatically generated by a lexical analyzer generator from descriptions of the patterns to be recognized. The descripions are in the form of regular expressions.

Lex is a lexical analyzer generator developed by M. E. Lesk and E. Schmidt of AT&T Bell Laboratories. The input to Lex is a file containing tokens defined using regular expressions. Lex produces an entire scanner module that can be compiled and linked to other compiler modules. A input file for Lex is of the form:

Lex generates a file containing the function `yylex()` which returns an integer denoting the token recognized.

```
C and scanner declarations
%%
Token definitions and actions
%%
C subroutines
```

The first section of the Lex file contains the C declaration to include the file (simple.tab.h) produced by Yacc/Bison which contains the definitions of the the multi-character tokens. The first section also contains Lex definitions used in the regular expressions. In this case, DIGIT is defined to be one of the symbols 0 through 9 and ID is defined to be a lower case letter followed by zero or more letters or digits.

```
%{
#include "Simple.tab.h" /* The tokens */
%}
```

```
DIGIT       [0-9]
ID          [a-z][a-z0-9]*
%%
Token definitions and actions
%%
C subroutines
```

The second section of the Lex file gives the regular expressions for each token to be recognized and a corresponding action. Strings of one or more digits are recognized as an integer and thus the value `INT` is returned to the parser. The reserved words of the language are strings of lower case letters (upper-case may be used but must be treated differently). Blanks, tabs and newlines are ignored. All other single character symbols are returned as themselves (the scanner places all input in the string `yytext`).

```
C and scanner declarations
%%
":="        { return(ASSGNOP);      }
{DIGIT}+    { return(NUMBER);       }
do          { return(DO);           }
else        { return(ELSE);         }
end         { return(END);          }
fi          { return(FI);           }
if          { return(IF);           }
in          { return(IN);           }
integer     { return(INTEGER);      }
let         { return(LET);          }
read        { return(READ);         }
skip        { return(SKIP);         }
then        { return(THEN);         }
while       { return(WHILE);        }
write       { return(WRITE);        }
{ID}        { return(IDENTIFIER);   }
[ \t\n]+ /* blank, tab, new line: eat up whitespace */
.           { return(yytext[0]);    }
%%
C subroutines
```

The values associated with the tokens are the integer values that the scanner returns to the parser upon recognizing the token.

Figure 3.1 gives the format of some of the regular expressions that may be used to define the tokens. There is a global variable `yylval` is accessible by both the scanner and the parser and is used to store additional information about the token.

The third section of the file is empty in this example but may contain C code associated with the actions.

Compiling the Lex file with the command lex *file*.lex (flex *file*.lex) results in the production of the file lex.yy.c which defines the C function yylex(). One each invocation, the function yylex() scans the input file an returns the next token.

. any character except newline

x match the character ‘x’

rs the regular expression r followed by the regular expression s; called “concatenation”

r|s either an r or an s

(r) match an r; parentheses are used to provide grouping.

r* zero or more r’s, where r is any regular expression

r+ one or more r’s

[xyz] a “character class”; in this case, the pattern matches either an ’x’, a ’y’, or a ‘z’.

[abj-oZ] a “character class” with a range in it; matches an ‘a’, a ‘b’, any letter from ‘j’ through ‘o’, or a ‘Z’.

{name} the expansion of the “name” definition.

X if X is an ‘a’, ‘b’, ‘f’, ‘n’, ‘r’, ‘t’, or ‘v’, then the ANSI-C interpretation of \x.

`"[+xyz]+\"+foo”` the literal string: [xyz]"foo

Figure 3.1: Lex/Flex Regular Expressions

Lex is distributed with the Unix operating system while Flex is a product of the Free Software Foundation, Inc.

For more information on using Lex/Flex consult the manual pages **lex**, **flex** and **flexdoc**, and see the paper LEX – Lexical Analyzer Generator by M. E. Lesk and E. Schmidt.