1. ***Structure of a Lex Specification***

   … definition section
   %%
   … rules section
   %%
   … user subroutines

2. ***Definition Section***

   2.1. *literal block*

   %{
   … C and C++ comments, directives, and declarations
   %}

   2.2. *definitions*

   A definition takes the form:

   NAME expression

   The name can contain letters, digits, and underscores, and must not start with a digit.

   In the rules section, patterns may include references to substitutions with the name in braces, for example, "{NAME}". The expression corresponding to the name is substituted literally into pattern. For example.

```
DIGIT                  [0-9]
…
%%
{DIGIT}+               process_integer();
{DIGIT}+\.{DIGIT}*  |
\.{DIGIT}+             process_real();
```

Figure 1. A lex specification that containing a definition.

3. ***Rules Section***

   A rule is a pattern followed by C or C++ code. For example:

```
%%
[ \t\n]+   ;
%%
```

Figure 2. A lex specification that discards white space**.**

3.1. Regular Expression Syntax

3.1.1. Metacharacters

Character	**Description**
.	Matches any single character except the newline character '\n'.

Character	**Description**
[]	Match any one of the characters with the brackets. A range of characters is indicated with the "**-**" (dash), e.g., "**[0-9]**" for any of the 10 digits. If the first character after the open bracket is a dash or a close bracket, it is not interpreted as a metacharacter If the first character is a circumflex "^" it changes the meaning to match any character except those within the brackets. (Such a character class *will* match a newline unless you explicitly exclude it.) Other metacharacters have no special meaning within square brackets except that C escape sequences starting with "\" are recognized.
*****	Matches zero or more of the preceding expression. For example, the pattern **a.*z** matches any string that starts with "a" and ends with "z", such as "az", "abz", or "alcatraz".
+	Matches one or more occurrence of the preceding regular expression. For example, **x+** matches "x", "xxx", or "xxxxx", but not an empty string, and **(ab)+** matches "ab", "abab", "ababab", and so forth.
?	Matches zero of one occurrence of the preceding regular expression. For example: **-?[0-9]+** indicates a whole number with an optional leading unary minus sign.
{}	A single number "{***n***}" means *n* repetitions of the preceding pattern, e.g., **[A-Z]{3}** matches any three upper case letters. If the braces contain two numbers separated by a comma, "{*n*,*m*}", they are a minimum and maximum number of repetitions of the preceding pattern. For example: **A{1,3}** matches one to three occurrences of the letter "**A**". If the second number is missing, it is taken to be infinite, so "{**1,**}" means the same as "**+**" and "**{0,}**" means the same as "*****".
\	If the following character is a lowercase letter, then it is a C escape sequence such as "**\t**" for tab. Some implementations also allow octal and hex characters in the form "**\123**" and "**\x3f**". Otherwise "\" quotes the following character, so"**\***" matches an asterisk.

Character	**Description**
()	Group a series of regular expressions together. Each of the "*****", "+", and "**[]**" effects only the expression immediately to its left, and "|" normally affects everything to its left and right. Parentheses can change this, for exa mple: **(ab|cd)?ef** matches "**abef**", "**cdef**", or just "|"
|	Match either the preceding regular expression or the subsequent regular expression. For example: **twelve|12** matches either "**twelve**" or "**12**"
"…"	Match everything withing the quotation marks literally. Metacharacters other than "\\" lose their meaning. For example: "/*" matches the two characters
/	Matches the preceding regular expression but only if followed by the following regular expression. For example: **0/1** matches "0" in the string "01" but does not match anything in the strings '**0**" or "**02**". Only one slash is permitted per pattern, and a pattern cannot contain both a slash and a trailing "**$**"
^	As the first character of a regular expression, it matches the beginning of a line; it is also used for negation within square brackets. Otherwise not special.
$	As the last character of a regular expression, it matches the end of a line – otherwise it is not special. The "**$**" has the same meaning as "**/\n**" when at the end of an expression.
<>	A name of list of names in angle brackets at the beginning of a pattern makes that pattern apply only in the given start states.

4. ***User Subroutines***
   User subroutines are C and C++ functions. Function prototypes must appear before their implementations in this section.

```
%{
#include <string>
#define ID      1
#define READ    2
#define WRITE   3
#define BEGAN   4
#define END     5
int TokenMgr(int t);
%}
%%
[ \t\n]+                ;
[a-z]+                  return TokenMgr(ID);
%%
int TokenMgr(int t)
{   string rw[]={“”,””,”read”,”write”,”begin”,”end”};
    for (int k=2;k<6;k++) if ((string)yytext==rw[k]) return k;
    return t;
}
```

Figure 2. A lex specification containing a user subroutine.