Lex Regular Expressions

Expression	Meaning
• **x**	The character "x"
• **"x"**	An "x", even if x is an operator.
• **\x**	An "x", even if x is an operator.
• **[xy]**	The character x or y.
• **[x-z]**	The character x, y, or z.
• **[^x]**	Any character but x.
• **.**	Any character but newline.
• **^x**	An x at the beginning of a line.
• **<y>x**	An x when Lex is in start condition y.
• **x$**	An x at the end of a line.
• **x?**	An optional x.
• **x***	0,1,2,... instances of x.
• **x+**	1,2,3,... instances of x.
• **x|y**	An x or a y.
• **(x)**	An x.
• **x/y**	An x, but only if followed by a y.
• **x{m,n}**	m through n occurrences of x
• **{xx}**	The xlation of xx from the definitions section

Lex Example: Building Symbol Tables

```
     /* definitions section */

%{
enum {
   LOOKUP = 0,
   NODE,
   INPUT,
   FINAL,
   START
};

short input_state;

short add_word     ( short type, char *word );
short lookup_word  ( char *word );
%}

%%

    /* rules section */

\n        { state = LOOKUP; }

^defnode  { state = NODE; }
^definput { state = INPUT; }
^deffinal { state = FINAL; }
^defstart { state = START; }
```

```
[a-zA-Z]+ {
    if (state != LOOKUP)
       add_word(state,yytext);
    else
    {
        switch(lookup_word(yytext))
        {
           /* read in name and do something */

           case NODE: ...
           case INPUT : ...
           .
           .
           .
           /* what if it's mispelt or
   undeclared? */

           default : printf ("%s: Unknown
   token!", yytext");
        }
    }
}

.     ;     /* anything else - just ignore */

%%

/* "main()" and "add_word" and "lookup_word",
   that would maintain lists of words and
   and search through them. As long as they do
   their job it doesn't matter what they look
   like. You can do that stuff right?  */
```

```
/* y.tab.h - file of token defns */
#define NODE 101
#define INPUT 102
#define FINAL 103
#define START 104
```

```
%{
#include "y.tab.h"    /* we need to know what
                        the tokens are */
#define LOOKUP   0    /* this is OK, Yacc
                        reserves 0, not Lex*/
short input_state;
short add_word     ( short type, char *word );
short lookup_word  ( char *word );
%}

%%
\n        { state = LOOKUP; }

^defnode  { state = NODE; }
^definput { state = INPUT; }
^deffinal { state = FINAL; }
^defstart { state = START; }
```

```
[a-zA-Z]+ {
    if (state != LOOKUP)
       add_word(state,yytext);
    else
    {
        switch(lookup_word(yytext))
        {
           /* read in name and if OK pass to
   Yacc */

           case NODE:
              return NODE;
              break;
           case INPUT :
              return INPUT;
              break;
           .
           .
       /* what if it's mispelt or undeclared? */
           default : printf ("%s: Unknown
                              token!", yytext");
        }
    }
}

\.\n    { input_state = LOOKUP; return 0; }  /*
                                        new rule!*/

.      ;     /* anything else - just ignore */

%%
           /* same as before here ... */
```