%s FOO

%%

[A-Z] { BEGIN(FOO); }

. {}

<FOO>. {}

<FOO>\n { BEGIN(INITIAL); }

%%

Here the initial state, i.e. the state INITIAL is the state that matches against the first two patterns, i.e. the "default" state. If you read any upper case character, you'll end up in state FOO. In state FOO, if you encounter a newline you will end up in the initial state, which is again the state that matches the first two rules.

To illustrate the uses of start conditions, here is a scanner which provides two different interpretations of a string like ‘123.456’. By default it will treat it as three tokens, the integer ‘123’, a dot (‘.’), and the integer ‘456’. But if the string is preceded earlier in the line by the string ‘expect-floats’ it will treat it as a single token, the floating-point number ‘123.456’:

%{

#include <math.h>

%}

%s expect

%%

expect-floats BEGIN(expect);

<expect>[0-9]+.[0-9]+ {

printf( "found a float, = %f\n",

atof( yytext ) );

}

<expect>\n {

/\* that's the end of the line, so

\* we need another "expect-number"

\* before we'll recognize any more

\* numbers

\*/

BEGIN(INITIAL);

}

[0-9]+ {

printf( "found an integer, = %d\n",

atoi( yytext ) );

}

"." printf( "found a dot\n" );

<https://www.cs.virginia.edu/~cr4bd/flex-manual/Start-Conditions.html>