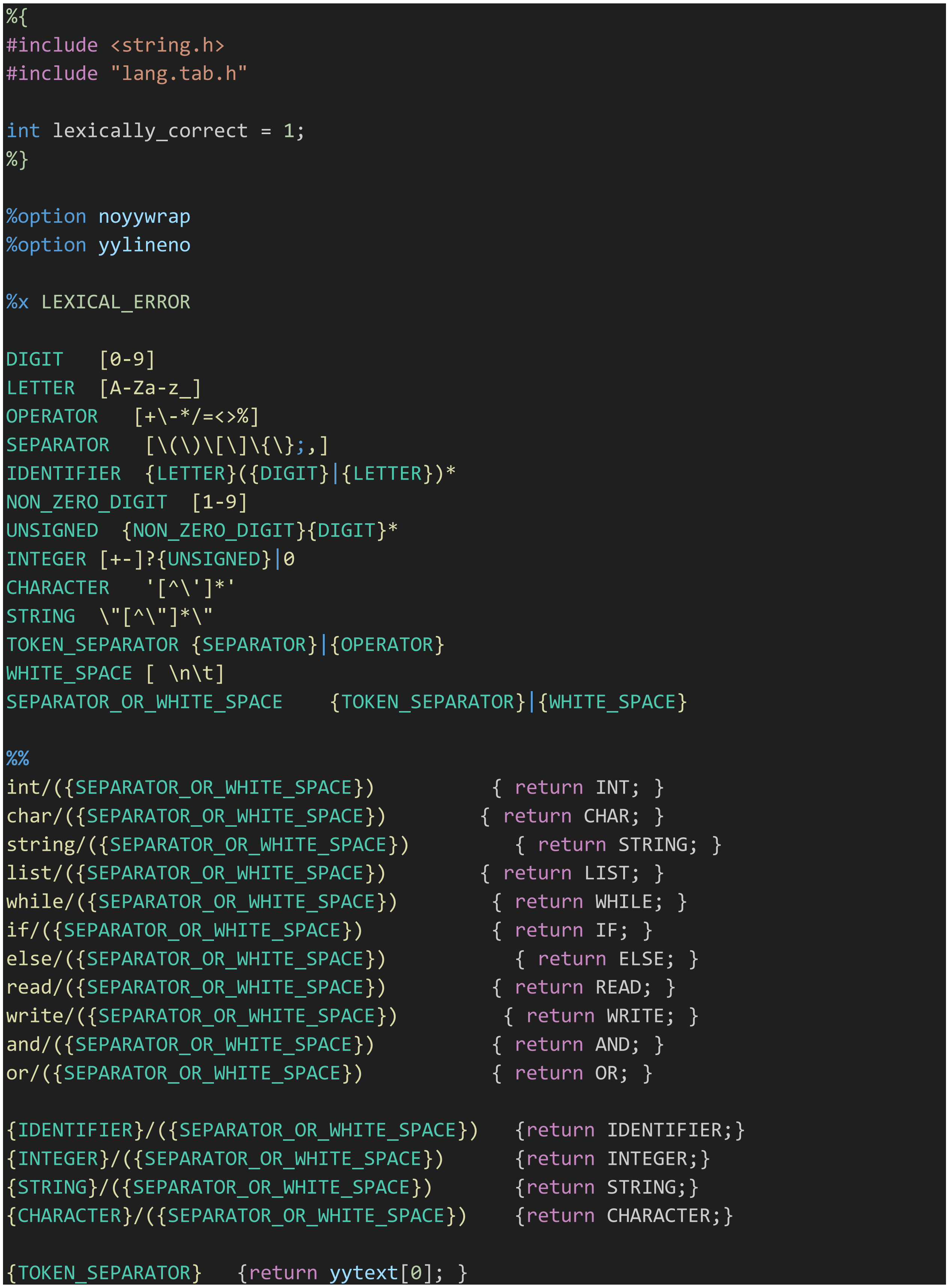

```
%{
#include <string.h>
#include "lang.tab.h"

int lexically_correct = 1;
%}

%option noyywrap
%option yylineno

%x LEXICAL_ERROR

DIGIT   [0-9]
LETTER  [A-Za-z_]
OPERATOR   [+\-*/=<>%]
SEPARATOR   [\(\)\[\]\{\};,]
IDENTIFIER  {LETTER}({DIGIT}|{LETTER})*
NON_ZERO_DIGIT  [1-9]
UNSIGNED  {NON_ZERO_DIGIT}{DIGIT}*
INTEGER [+-]?{UNSIGNED}|0
CHARACTER   '[^\']*'
STRING  \"[^\"]*\"
TOKEN_SEPARATOR {SEPARATOR}|{OPERATOR}
WHITE_SPACE [ \n\t]
SEPARATOR_OR_WHITE_SPACE    {TOKEN_SEPARATOR}|{WHITE_SPACE}

%%
int/({SEPARATOR_OR_WHITE_SPACE})          { return INT; }
char/({SEPARATOR_OR_WHITE_SPACE})        { return CHAR; }
string/({SEPARATOR_OR_WHITE_SPACE})         { return STRING; }
list/({SEPARATOR_OR_WHITE_SPACE})        { return LIST; }
while/({SEPARATOR_OR_WHITE_SPACE})        { return WHILE; }
if/({SEPARATOR_OR_WHITE_SPACE})           { return IF; }
else/({SEPARATOR_OR_WHITE_SPACE})           { return ELSE; }
read/({SEPARATOR_OR_WHITE_SPACE})         { return READ; }
write/({SEPARATOR_OR_WHITE_SPACE})         { return WRITE; }
and/({SEPARATOR_OR_WHITE_SPACE})          { return AND; }
or/({SEPARATOR_OR_WHITE_SPACE})           { return OR; }

{IDENTIFIER}/({SEPARATOR_OR_WHITE_SPACE})   {return IDENTIFIER;}
{INTEGER}/({SEPARATOR_OR_WHITE_SPACE})      {return INTEGER;}
{STRING}/({SEPARATOR_OR_WHITE_SPACE})       {return STRING;}
{CHARACTER}/({SEPARATOR_OR_WHITE_SPACE})    {return CHARACTER;}

{TOKEN_SEPARATOR}   {return yytext[0]; }
```

```
"=="   {return EQ;}
"<="   {return LE;}
">="   {return GE;}
"!="    {return NOTEQ;}

{WHITE_SPACE}+  /* eat up whitespace */

. {
    BEGIN(LEXICAL_ERROR);
    yymore();
}

<LEXICAL_ERROR>([^+\-*/=<>%!\(\)\[\]\{\};,\n\t
])*/({TOKEN_SEPARATOR}|{WHITE_SPACE}) {
    printf("Lexical error on line %d, token \"%s\" is not an reserved word,
operator, separator, identifier or constant\n", yylineno, yytext);
    lexically_correct = 0;
    BEGIN(INITIAL);
}
```