Proj2 lex file

```
%{
  #include "y.tab.h"
   int currLine = 1, currPos = 1;
%}

DIGIT    [0-9]
UNDERSCORE [_]
LETTER [a-zA-Z]

%%

"function"      {currPos += yyleng; return FUNCTION;}
"beginparams"   {currPos += yyleng; return BEGIN_PARAMS;}
"endparams"     {currPos += yyleng; return END_PARAMS;}
"beginlocals"   {currPos += yyleng; return BEGIN_LOCALS;}
"endlocals"     {currPos += yyleng; return END_LOCALS;}
"beginbody"     {currPos += yyleng; return BEGIN_BODY;}
"endbody"       {currPos += yyleng; return END_BODY;}
"integer"       {currPos += yyleng; return INTEGER;}
"array"         {currPos += yyleng; return ARRAY;}
"of"            {currPos += yyleng; return OF;}
"if"            {currPos += yyleng; return IF;}
"then"          {currPos += yyleng; return THEN;}
"endif"         {currPos += yyleng; return ENDIF;}
"else"          {currPos += yyleng; return ELSE;}
"while"         {currPos += yyleng; return WHILE;}
"do"            {currPos += yyleng; return DO;}
"for"           {currPos += yyleng; return FOR;}
"beginloop"     {currPos += yyleng; return BEGINLOOP;}
"endloop"       {currPos += yyleng; return ENDLOOP;}
"continue"      {currPos += yyleng; return CONTINUE;}
"read"          {currPos += yyleng; return READ;}
"write"         {currPos += yyleng; return WRITE;}
"and"           {currPos += yyleng; return AND;}
"or"            {currPos += yyleng; return OR;}
"not"           {currPos += yyleng; return NOT;}
"true"          {currPos += yyleng; return TRUE;}
"false"         {currPos += yyleng; return FALSE;}
"return"        {currPos += yyleng; return RETURN;}

"-"             {currPos += yyleng; return SUB;}
"+"             {currPos += yyleng; return ADD;}
"*"             {currPos += yyleng; return MULT;}
"/"             {currPos += yyleng; return DIV;}
"%"             {currPos += yyleng; return MOD;}
"=="            {currPos += yyleng; return EQ;}
"<>"            {currPos += yyleng; return NEQ;}
```

```
"<"             {currPos += yyleng; return LT;}
">"             {currPos += yyleng; return GT;}
"<="            {currPos += yyleng; return LTE;}
">="            {currPos += yyleng; return GTE;}
";"             {currPos += yyleng; return SEMICOLON;}
":"             {currPos += yyleng; return COLON;}
","             {currPos += yyleng; return COMMA;}
"("             {currPos += yyleng; return L_PAREN;}
")"             {currPos += yyleng; return R_PAREN;}
"["             {currPos += yyleng; return L_SQUARE_BRACKET;}
"]"             {currPos += yyleng; return R_SQUARE_BRACKET;}
":="            {currPos += yyleng; return ASSIGN;}

{DIGIT}+       {yylval.ival=atoi(strdup(yytext)); currPos += yyleng; return NUMBER;}
{LETTER}+(({DIGIT}|{UNDERSCORE}|{LETTER})*({DIGIT}|{LETTER})+)*     {yylval.cval=strdup(yy
text); currPos += yyleng; return IDENT;}

[ \t]+         {/* ignore spaces */ currPos += yyleng;}

"\n"           {currLine++; currPos = 1;}
"##".*           {currLine++; currPos = 1;}

.              {printf("Error at line %d, column %d: unrecognized symbol \"%s\"\n", currLi
ne, currPos, yytext);}

{DIGIT}+({DIGIT}|{UNDERSCORE}|{LETTER})*              {printf("Error at line %d, column %
d: identifier \"%s\" must begin with a letter \n", currLine, currPos, yytext);}

{LETTER}+({DIGIT}|{UNDERSCORE}|{LETTER})*{UNDERSCORE}+             {printf("Error at line
 %d, column %d: identifier \"%s\" cannot end with an underscore \n", currLine, currPos, yy
text);}

%%
/*
int main(int argc, char ** argv)
{
   if(argc >= 2)
   {
      yyin = fopen(argv[1], "r");
      if(yyin == NULL)
      {
      }
   }
  else
    {
      yyin = stdin;
   }
}
*/
```