Proj3 lex file

```
%{
#include <iostream>
#define YY_DECL yy::parser::symbol_type yylex()
#include "parser.tab.hh"

static yy::location loc;
%}

%option noyywrap

%{
#define YY_USER_ACTION loc.columns(yyleng);
%}

  /* your definitions here */
DIGIT    [0-9]
UNDERSCORE [_]
LETTER [a-zA-Z]

  /* your definitions end */

%%

%{
loc.step();
%}

  /* your rules here */

  /* use this structure to pass the Token :
   * return yy::parser::make_TokenName(loc)
   * if the token has a type you can pass it's value
   * as the first argument. as an example we put
   * the rule to return token function.
   */
"function"      {return yy::parser::make_FUNCTION(loc);}
"beginparams"   {return yy::parser::make_BEGIN_PARAMS(loc);}
"endparams"    {return yy::parser::make_END_PARAMS(loc);}
"beginlocals"   {return yy::parser::make_BEGIN_LOCALS(loc);}
"endlocals"     {return yy::parser::make_END_LOCALS(loc);}
"beginbody"     {return yy::parser::make_BEGIN_BODY(loc);}
"endbody"       {return yy::parser::make_END_BODY(loc);}
"integer"       {return yy::parser::make_INTEGER(loc);}
"array"         {return yy::parser::make_ARRAY(loc);}
"of"            {return yy::parser::make_OF(loc);}
"if"            {return yy::parser::make_IF(loc);}
"then"          {return yy::parser::make_THEN(loc);}
```

```
"endif"         {return yy::parser::make_ENDIF(loc);}
"else"          {return yy::parser::make_ELSE(loc);}
"while"         {return yy::parser::make_WHILE(loc);}
"do"            {return yy::parser::make_DO(loc);}
"for"           {return yy::parser::make_FOR(loc);}
"beginloop"     {return yy::parser::make_BEGINLOOP(loc);}
"endloop"       {return yy::parser::make_ENDLOOP(loc);}
"continue"      {return yy::parser::make_CONTINUE(loc);}
"read"          {return yy::parser::make_READ(loc);}
"write"         {return yy::parser::make_WRITE(loc);}
"and"           {return yy::parser::make_AND(loc);}
"or"            {return yy::parser::make_OR(loc);}
"not"           {return yy::parser::make_NOT(loc);}
"true"          {return yy::parser::make_TRUE(loc);}
"false"         {return yy::parser::make_FALSE(loc);}
"return"        {return yy::parser::make_RETURN(loc);}

"-"             {return yy::parser::make_SUB(loc);}
"+"             {return yy::parser::make_ADD(loc);}
"*"             {return yy::parser::make_MULT(loc);}
"/"             {return yy::parser::make_DIV(loc);}
"%"             {return yy::parser::make_MOD(loc);}
"=="            {return yy::parser::make_EQ(loc);}
"<>"            {return yy::parser::make_NEQ(loc);}
"<"             {return yy::parser::make_LT(loc);}
">"             {return yy::parser::make_GT(loc);}
"<="            {return yy::parser::make_LTE(loc);}
">="            {return yy::parser::make_GTE(loc);}
";"             {return yy::parser::make_SEMICOLON(loc);}
":"             {return yy::parser::make_COLON(loc);}
","             {return yy::parser::make_COMMA(loc);}
"("             {return yy::parser::make_L_PAREN(loc);}
")"             {return yy::parser::make_R_PAREN(loc);}
"["             {return yy::parser::make_L_SQUARE_BRACKET(loc);}
"]"             {return yy::parser::make_R_SQUARE_BRACKET(loc);}
":="            {return yy::parser::make_ASSIGN(loc);}

{DIGIT}+       {return yy::parser::make_NUMBER(stoi(yytext),loc);}
{LETTER}+(({DIGIT}|{UNDERSCORE}|{LETTER})*({DIGIT}|{LETTER})+)*     {return yy::parser::ma
ke_IDENT(yytext,loc);}

[ \t]+         {}

"\n"           {}
"##".*           {}

.              {/*error at line and column*/}

{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 lin
e %d, column %d: identifier \"%s\" cannot end with an underscore \n", currLine, currPos, y
ytext);}*/}
```

```
 <<EOF>>  {return yy::parser::make_END(loc);}
  /* your rules end */

%%
```