CSPC62 - Compilers
Lab 3

Topic - Working example of lexical analysis for your own language

Team Members

Roll No. - 106119029 **Name -** Dipesh Kaphle
Roll No. - 106119073 **Name -** Marmik Upadhyay
Roll No. - 106119117 **Name -** Sobhagya Singh Dewal

Code:

Lexer.l

```
%{
#include <string>
#include <iostream>
#include <vector>
#include <unordered_map>
#include "colors.h"
extern FILE* yyin;
struct Loc{
   int col =0;
   int line=0;
};
enum TokenType{
TLET,
TBOOL,
TIF,
TELSE,
TWHILE,
TDOT,
TID,
TNUM,
TFN,
TLPAREN,
TRPAREN,
TLCURLY,
TRCURLY,
TSEMICOLON,
TCOLON,
TDATA,
TEQUAL,
TCEQ,
TCNEQ,
TCLT,
TCLE,
TCGT,
TCGE,
TPLUS,
```

```
TMINUS,
TMUL,
TDIV,
TAND,
TOR,
TNOT
};


#define TOK(t) Token(t,std::string(yytext, yyleng))
#define INC l.col+=yyleng
Loc l;

struct Token{
   TokenType type;
   std::string lit;
   Loc location;
   Token(TokenType to, std::string literal): type(to),lit(literal),
location(l){}
};


std::vector<Token> tokens;
std::unordered_map<TokenType,std::string> tokenMap = {
   {TLET, "TLET"},
   {TBOOL, "TBOOL"},
   {TIF, "TIF"},
   {TELSE, "TELSE"},
   {TWHILE, "TWHILE"},
   {TDOT, "TDOT"},
   {TID, "TID"},
   {TNUM, "TNUM"},
   {TFN, "TFN"},
   {TLPAREN, "TLPAREN"},
   {TRPAREN, "TRPAREN"},
   {TLCURLY, "TLCURLY"},
   {TRCURLY, "TRCURLY"},
   {TSEMICOLON, "TSEMICOLON"},
   {TCOLON, "TCOLON"},
   {TDATA, "TDATA"},
```

```
   {TEQUAL, "TEQUAL"},
   {TCEQ, "TCEQ"},
   {TCNEQ, "TCNEQ"},
   {TCLT, "TCLT"},
   {TCLE, "TCLE"},
   {TCGT, "TCGT"},
   {TCGE, "TCGE"},
   {TPLUS, "TPLUS"},
   {TMINUS, "TMINUS"},
   {TMUL, "TMUL"},
   {TDIV, "TDIV"},
   {TAND, "TAND"},
   {TOR, "TOR"},
   {TNOT, "TNOT"}
};

extern "C" int yywrap() { return 1;}
bool hasError = false;
%}

%%

[ \t]                   { l.col++;}
[\n]                    { l.line++; l.col=0;}
"let"                   { INC; tokens.push_back(TOK(TLET));}
"data"                  { INC; tokens.push_back(TOK(TDATA));}
"true"                  { INC; tokens.push_back(TOK(TBOOL));}
"false"                 { INC; tokens.push_back(TOK(TBOOL));}
"if"                    { INC; tokens.push_back(TOK(TIF));}
"else"                  { INC; tokens.push_back(TOK(TELSE));}
"while"                 { INC; tokens.push_back(TOK(TWHILE));}
"fn"                    { INC; tokens.push_back(TOK(TFN));}
"."                     { INC; tokens.push_back(TOK(TDOT));}
[_a-zA-z][_a-zA-Z0-9']* { INC; tokens.push_back(TOK(TID));}
[0-9]+\.[0-9]*          { INC; tokens.push_back(TOK(TNUM));}
[0-9]+                  { INC; tokens.push_back(TOK(TNUM));}
"("                     { INC; tokens.push_back(TOK(TLPAREN));}
")"                     { INC; tokens.push_back(TOK(TRPAREN));}
"{"                     { INC; tokens.push_back(TOK(TLCURLY));}
"}"                     { INC; tokens.push_back(TOK(TRCURLY));}
```

```
";"                     { INC; tokens.push_back(TOK(TSEMICOLON));}
":"                     { INC; tokens.push_back(TOK(TCOLON));}
"="                     { INC; tokens.push_back(TOK(TEQUAL));}
"=="                    { INC; tokens.push_back(TOK(TCEQ));}
"!="                    { INC; tokens.push_back(TOK(TCNEQ));}
"!"                     { INC; tokens.push_back(TOK(TNOT)); }
"<"                     { INC; tokens.push_back(TOK(TCLT));}
"<="                    { INC; tokens.push_back(TOK(TCLE));}
">"                     { INC; tokens.push_back(TOK(TCGT));}
">="                    { INC; tokens.push_back(TOK(TCGE));}
"+"                     { INC; tokens.push_back(TOK(TPLUS));}
"-"                     { INC; tokens.push_back(TOK(TMINUS));}
"*"                     { INC; tokens.push_back(TOK(TMUL));}
"/"                     { INC; tokens.push_back(TOK(TDIV));}
"&&"                    { INC; tokens.push_back(TOK(TAND));}
"||"                    { INC; tokens.push_back(TOK(TOR));}
.                       { std::cerr << RED << "Unknown input " << BOLDRED <<
std::string(yytext,yyleng) << " at Line : " << l.line << " and Col : " <<
l.col << RESET << std::endl;
                            hasError = true;
                            yyterminate();
                        }
%%

int main(){
   yyin = fopen("input.txt","r");
   yylex();
   if(hasError) {
       std::cerr << RED << "Error in parsing" << RESET << std::endl;
       return 1;
   }
   bool show_lines=true;
   for(auto &x: tokens){
       std::cout<< BOLDWHITE << "Lexeme: "<< BOLDMAGENTA << x.lit << WHITE
<<", Token: " << BOLDMAGENTA << tokenMap.at(x.type) << RESET;
       if(show_lines){
           std::cout << " [at line: "<< CYAN << x.location.line << RESET <<
" and col: " << CYAN << x.location.col << RESET << "]\n";
       }else{
           std::cout<<"\n"; } } }
```

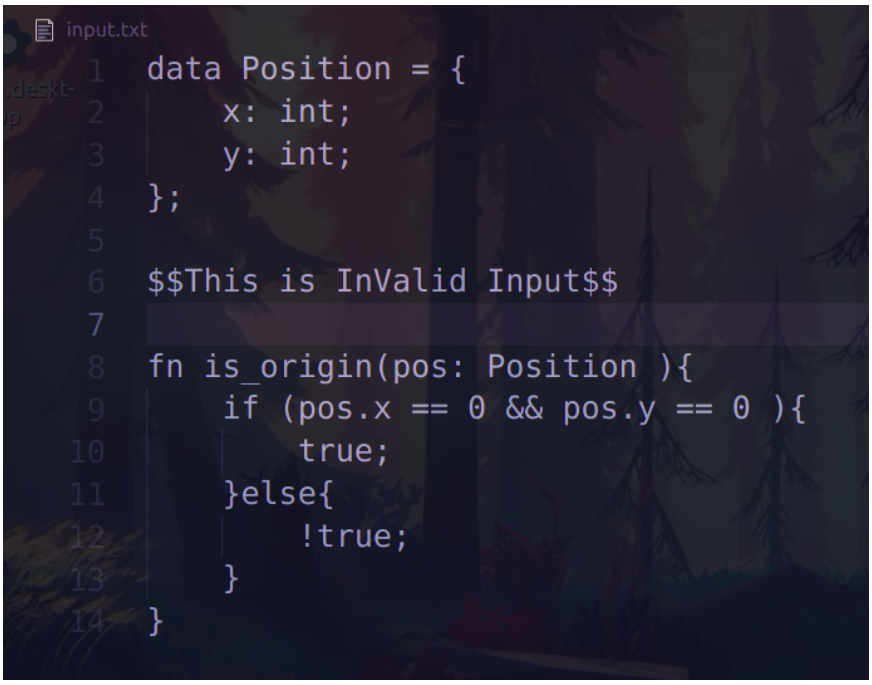
Colors.h (for pretty printing the output)

```
#ifndef _WIN32
#define RESET "\033[0m"
#define BLACK "\033[30m"              /* Black */
#define RED "\033[31m"                /* Red */
#define GREEN "\033[32m"              /* Green */
#define YELLOW "\033[33m"             /* Yellow */
#define BLUE "\033[34m"               /* Blue */
#define MAGENTA "\033[35m"            /* Magenta */
#define CYAN "\033[36m"               /* Cyan */
#define WHITE "\033[37m"              /* White */
#define BOLDBLACK "\033[1m\033[30m"   /* Bold Black */
#define BOLDRED "\033[1m\033[31m"     /* Bold Red */
#define BOLDGREEN "\033[1m\033[32m"   /* Bold Green */
#define BOLDYELLOW "\033[1m\033[33m"  /* Bold Yellow */
#define BOLDBLUE "\033[1m\033[34m"    /* Bold Blue */
#define BOLDMAGENTA "\033[1m\033[35m" /* Bold Magenta */
#define BOLDCYAN "\033[1m\033[36m"    /* Bold Cyan */
#define BOLDWHITE "\033[1m\033[37m"   /* Bold White */
#define UNDERLINE "\033[4m"
#endif
#ifdef _WIN32
#define RESET ""
#define BLACK ""
#define RED ""
#define GREEN ""
#define YELLOW ""
#define BLUE ""
#define MAGENTA ""
#define CYAN ""
#define WHITE ""
#define BOLDBLACK ""
#define BOLDRED ""
#define BOLDGREEN ""
#define BOLDYELLOW ""
#define BOLDBLUE ""
#define BOLDMAGENTA ""
#define BOLDCYAN ""
#define BOLDWHITE ""
#define UNDERLINE ""
```

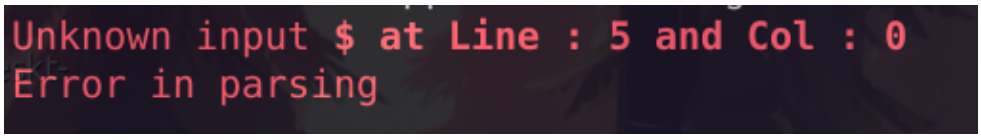
```
#endif
```

Output

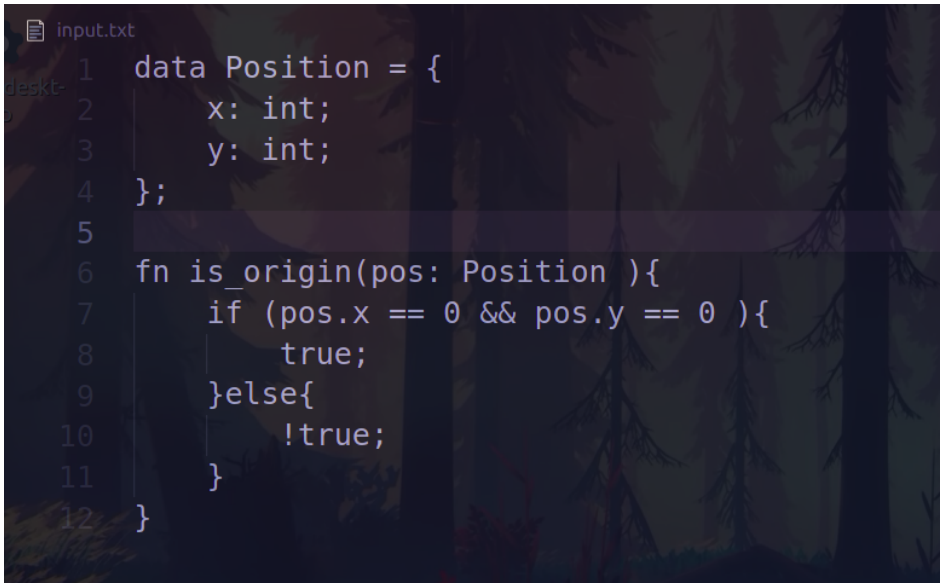
Input (With Error)



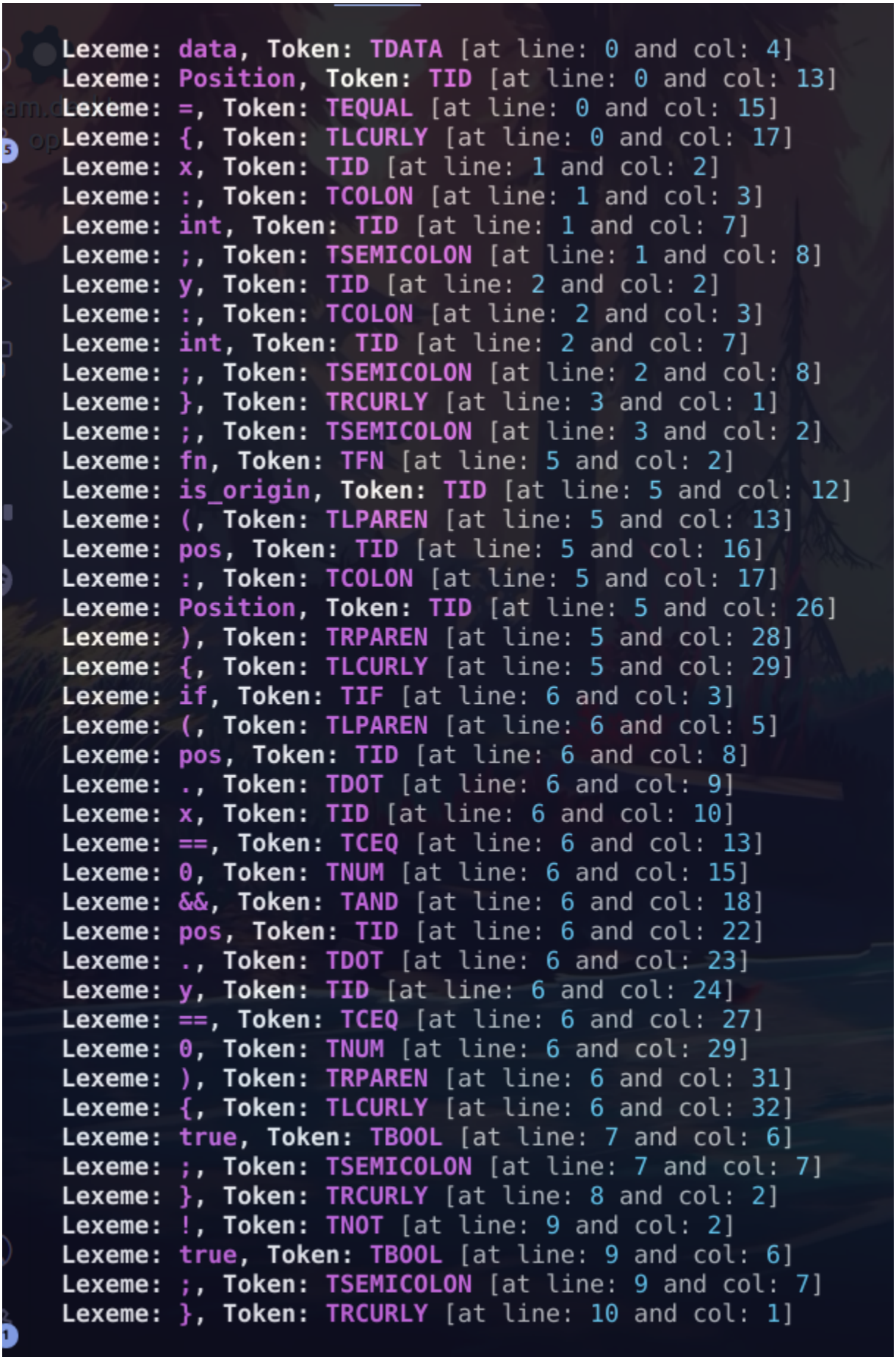
Output (With Error)



Input (Without Error)



Output (Without Error)



```
Lexeme: data, Token: TDATA [at line: 0 and col: 4]
Lexeme: Position, Token: TID [at line: 0 and col: 13]
Lexeme: =, Token: TEQUAL [at line: 0 and col: 15]
Lexeme: {, Token: TLCURLY [at line: 0 and col: 17]
Lexeme: x, Token: TID [at line: 1 and col: 2]
Lexeme: :, Token: TCOLON [at line: 1 and col: 3]
Lexeme: int, Token: TID [at line: 1 and col: 7]
Lexeme: ;, Token: TSEMICOLON [at line: 1 and col: 8]
Lexeme: y, Token: TID [at line: 2 and col: 2]
Lexeme: :, Token: TCOLON [at line: 2 and col: 3]
Lexeme: int, Token: TID [at line: 2 and col: 7]
Lexeme: ;, Token: TSEMICOLON [at line: 2 and col: 8]
Lexeme: }, Token: TRCURLY [at line: 3 and col: 1]
Lexeme: ;, Token: TSEMICOLON [at line: 3 and col: 2]
Lexeme: fn, Token: TFN [at line: 5 and col: 2]
Lexeme: is_origin, Token: TID [at line: 5 and col: 12]
Lexeme: (, Token: TLPAREN [at line: 5 and col: 13]
Lexeme: pos, Token: TID [at line: 5 and col: 16]
Lexeme: :, Token: TCOLON [at line: 5 and col: 17]
Lexeme: Position, Token: TID [at line: 5 and col: 26]
Lexeme: ), Token: TRPAREN [at line: 5 and col: 28]
Lexeme: {, Token: TLCURLY [at line: 5 and col: 29]
Lexeme: if, Token: TIF [at line: 6 and col: 3]
Lexeme: (, Token: TLPAREN [at line: 6 and col: 5]
Lexeme: pos, Token: TID [at line: 6 and col: 8]
Lexeme: ., Token: TDOT [at line: 6 and col: 9]
Lexeme: x, Token: TID [at line: 6 and col: 10]
Lexeme: ==, Token: TCEQ [at line: 6 and col: 13]
Lexeme: 0, Token: TNUM [at line: 6 and col: 15]
Lexeme: &&, Token: TAND [at line: 6 and col: 18]
Lexeme: pos, Token: TID [at line: 6 and col: 22]
Lexeme: ., Token: TDOT [at line: 6 and col: 23]
Lexeme: y, Token: TID [at line: 6 and col: 24]
Lexeme: ==, Token: TCEQ [at line: 6 and col: 27]
Lexeme: 0, Token: TNUM [at line: 6 and col: 29]
Lexeme: ), Token: TRPAREN [at line: 6 and col: 31]
Lexeme: {, Token: TLCURLY [at line: 6 and col: 32]
Lexeme: true, Token: TBOOL [at line: 7 and col: 6]
Lexeme: ;, Token: TSEMICOLON [at line: 7 and col: 7]
Lexeme: }, Token: TRCURLY [at line: 8 and col: 2]
Lexeme: !, Token: TNOT [at line: 9 and col: 2]
Lexeme: true, Token: TBOOL [at line: 9 and col: 6]
Lexeme: ;, Token: TSEMICOLON [at line: 9 and col: 7]
Lexeme: }, Token: TRCURLY [at line: 10 and col: 1]
```

~Thank You~