%option noyywrap

%option never-interactive

%option c++

%option yylineno

%x CONST\_STRING

%x MULTILINE\_COMMENT

%x ONELINE\_COMMENT

%x CHAR

DIGIT [0-9]

DECIMAL\_NUMBER ({DIGIT}+\_)\*{DIGIT}+

HEXIT [0-9a-fA-F]

HEX\_NUMBER 0[xX]({HEXIT}+\_)\*{HEXIT}+

OCTIT [0-7]

OCT\_NUMBER 0[cCoO]({OCTIT}+\_)\*{OCTIT}+

BINIT [01]

BIN\_NUMBER 0[bB]({BINIT}+\_)\*{BINIT}+

REAL\_NUMBER {DECIMAL\_NUMBER}?\.{DECIMAL\_NUMBER}|{DECIMAL\_NUMBER}\.{DECIMAL\_NUMBER}?

REAL\_NUMBER\_EXPONENT ({REAL\_NUMBER}|{DECIMAL\_NUMBER})[eE][\-+]?{DECIMAL\_NUMBER}

%{

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdint.h>

#include "ParseNumbers.h"

#pragma warning(disable : 4996)

char buffer[1000];

%}

%%

%{

int64\_t int\_number;

double real\_number;

int int\_buffer;

char buf\_char[2];

%}

abstract { printf("(%s) - Found KW\_ABSTRACT\n", yytext); }

val { printf("(%s) - Found KW\_VAL\n", yytext); }

case { printf("(%s) - Found KW\_CASE\n", yytext); }

catch { printf("(%s) - Found KW\_CATCH\n", yytext); }

class { printf("(%s) - Found KW\_CLASS\n", yytext); }

def { printf("(%s) - Found KW\_DEF\n", yytext); }

do { printf("(%s) - Found KW\_do\n", yytext); }

else { printf("(%s) - Found KW\_ELSE\n", yytext); }

extends { printf("(%s) - Found KW\_EXTENDS\n", yytext); }

false { printf("(%s) - Found KW\_FALSE\n", yytext); }

for { printf("(%s) - Found KW\_FOR\n", yytext); }

final { printf("(%s) - Found KW\_FINAL\n", yytext); }

finally { printf("(%s) - Found KW\_FINALLY\n", yytext); }

forSome { printf("(%s) - Found KW\_forSome\n", yytext); }

if { printf("(%s) - Found KW\_IF\n", yytext); }

implicit { printf("(%s) - Found KW\_IMPLICIT\n", yytext); }

import { printf("(%s) - Found KW\_IMPORT\n", yytext); }

lazy { printf("(%s) - Found KW\_LAZY\n", yytext); }

match { printf("(%s) - Found KW\_MATCH\n", yytext); }

new { printf("(%s) - Found KW\_NEW\n", yytext); }

null { printf("(%s) - Found KW\_NULL\n", yytext); }

object { printf("(%s) - Found KW\_OBJECT\n", yytext); }

override { printf("(%s) - Found KW\_OVERRIDE\n", yytext); }

package { printf("(%s) - Found KW\_PACKAGE\n", yytext); }

private { printf("(%s) - Found KW\_PRIVATE\n", yytext); }

protected { printf("(%s) - Found KW\_PROTECTED\n", yytext); }

print { printf("(%s) - Found KW\_PRINT\n", yytext); }

println { printf("(%s) - Found KW\_PRINTLN\n", yytext); }

return { printf("(%s) - Found KW\_RETURN\n", yytext); }

sealed { printf("(%s) - Found KW\_SEALED\n", yytext); }

super { printf("(%s) - Found KW\_SUPER\n", yytext); }

this { printf("(%s) - Found KW\_THIS\n", yytext); }

throw { printf("(%s) - Found KW\_THROW\n", yytext); }

trait { printf("(%s) - Found KW\_TRAIT\n", yytext); }

true { printf("(%s) - Found KW\_TRUE\n", yytext); }

try { printf("(%s) - Found KW\_TRY\n", yytext); }

type { printf("(%s) - Found KW\_TYPE\n", yytext); }

var { printf("(%s) - Found KW\_VAR\n", yytext); }

while { printf("(%s) - Found KW\_WHILE\n", yytext); }

with { printf("(%s) - Found KW\_WITH\n", yytext); }

yield { printf("(%s) - Found KW\_YIELD\n", yytext); }

\" {buffer[0]=0; BEGIN(CONST\_STRING);}

<CONST\_STRING>[^\\\"]+ {strcat(buffer,yytext);}

<CONST\_STRING>\\\n[\n\t\r ]\* {strcat(buffer, "");}

<CONST\_STRING>\\\\ {strcat(buffer, "\\");}

<CONST\_STRING>\\n {strcat(buffer, "\n");}

<CONST\_STRING>\\t {strcat(buffer, "\t");}

<CONST\_STRING>\\0 {strcat(buffer, "\0");}

<CONST\_STRING>\\r {strcat(buffer, "\r");}

<CONST\_STRING>\\\" {strcat(buffer, "\"");}

<CONST\_STRING>\\\' {strcat(buffer, "\'");}

<CONST\_STRING>\\x[0-7][0-9A-Fa-f] { buf\_char[0] = strtol(yytext+2, NULL, 16); buf\_char[1] = 0; strcat(buffer, buf\_char);}

<CONST\_STRING><<EOF>> {printf("(%s) - Found ERROR\_STRING\n", buffer); BEGIN(INITIAL);}

<CONST\_STRING>\\ {printf("(%s) - Found NOT\_ESCAPED\_SLASH\n", buffer); BEGIN(INITIAL);}

<CONST\_STRING>\" {printf("(%s) - Found CONST\_STRING\n", buffer); BEGIN(INITIAL);}

\' {buffer[0]=0; BEGIN(CHAR);}

<CHAR>\\x[0-7][0-9A-Fa-f]\' { yytext[strlen(yytext)-1] = '\0'; buf\_char[0] = strtol(yytext+2, NULL, 16); printf( "(%c) - Found CHAR\n", buf\_char[0] ); BEGIN(INITIAL);}

<CHAR>[^\\\']+ {strcat(buffer,yytext);}

<CHAR>\\\\ {strcat(buffer, "\\");}

<CHAR>\\n {strcat(buffer, "\n");}

<CHAR>\\t {strcat(buffer, "\t");}

<CHAR>\\0 {strcat(buffer, "\0");}

<CHAR>\\r {strcat(buffer, "\r");}

<CHAR>\\\" {strcat(buffer, "\"");}

<CHAR>\\\' {strcat(buffer, "\'");}

<CHAR><<EOF>> {printf("(%s) - Error CHAR\n", buffer); BEGIN(INITIAL);}

<CHAR>\' {

if(strlen(buffer)==1){

printf("(%s) - Found CHAR\n", buffer); BEGIN(INITIAL);

}else{

printf("(%s) - Error CHAR\n", buffer); BEGIN(INITIAL);

}

}

\/\\* {buffer[0]=0; BEGIN(MULTILINE\_COMMENT);}

<MULTILINE\_COMMENT>[^\\*\/]+ {strcat(buffer,yytext);}

<MULTILINE\_COMMENT>\\* {strcat(buffer, "\*");}

<MULTILINE\_COMMENT>\/ {strcat(buffer, "/");}

<MULTILINE\_COMMENT><<EOF>> { printf("(%s) - Error COMMENT\n", buffer); BEGIN(INITIAL);}

<MULTILINE\_COMMENT>\\*\/ {printf("(%s) - Found MULTILINE\_COMMENT\n", buffer); BEGIN(INITIAL);}

\/\/ {buffer[0]=0; BEGIN(ONELINE\_COMMENT);}

<ONELINE\_COMMENT>[^\n] {strcat(buffer, yytext);}

<ONELINE\_COMMENT>\n {printf("(%s) - Found ONELINE\_COMMENT\n", buffer); BEGIN(INITIAL);}

<ONELINE\_COMMENT><<EOF>> {printf("(%s) - Found ONELINE\_COMMENT\n", buffer); BEGIN(INITIAL);}

\+ { printf("(%s) - Found PLUS\_OPERATOR\n", yytext); }

\- { printf("(%s) - Found MINUS\_OPERATOR\n", yytext); }

\\* { printf("(%s) - Found MUL\_OPERATOR\n", yytext); }

\% { printf("(%s) - Found MOD\_OPERATOR\n", yytext); }

\%\= { printf("(%s) - Found MOD\_ASSIGNMENT\_OPERATOR\n", yytext); }

\\*\\* { printf("(%s) - Found EXP\_OPERATOR\n", yytext); }

\\*\\*\= { printf("(%s) - Found EXP\_ASSIGNMENT\_OPERATOR\n", yytext); }

\/ { printf("(%s) - Found DIV\_OPERATOR\n", yytext); }

\+\= { printf("(%s) - Found PLUS\_ASSIGNMENT\_OPERATOR\n", yytext); }

\-\= { printf("(%s) - Found MINUS\_ASSIGNMENT\_OPERATOR\n", yytext); }

\\*\= { printf("(%s) - Found MUL\_ASSIGNMENT\_OPERATOR\n", yytext); }

\/\= { printf("(%s) - Found DIV\_ASSIGNMENT\_OPERATOR\n", yytext); }

\<= { printf("(%s) - Found LESS\_OR\_EQUAL\_OPERATOR\n", yytext); }

\<\<\= { printf("(%s) - Found LEFTSHIFT\_ASSIGNMENT\_OPERATOR\n", yytext); }

\>\>\= { printf("(%s) - Found RIGHTSHIFT\_ASSIGNMENT\_OPERATOR\n", yytext); }

\>= { printf("(%s) - Found MORE\_OR\_EQUAL\_OPERATOR\n", yytext); }

\< { printf("(%s) - Found LESS\_OPERATOR\n", yytext); }

\> { printf("(%s) - Found MORE\_OPERATOR\n", yytext); }

\!= { printf("(%s) - Found NOT\_EQUAL\_OPERATOR\n", yytext); }

= { printf("(%s) - Found ASSIGNMENT\_OPERATOR\n", yytext); }

\^ { printf("(%s) - Found CIRCUMFLEX\n", yytext); }

== { printf("(%s) - Found EQUAL\_OPERATOR\n", yytext); }

\! { printf("(%s) - Found NOT\_OPERATOR\n", yytext); }

\|\| { printf("(%s) - Found OR\_OPERATOR\n", yytext); }

\&\& { printf("(%s) - Found AND\_OPERATOR\n", yytext); }

\&\= { printf("(%s) - Found BITWISE\_AND\_ASSIGNMENT\_OPERATOR\n", yytext); }

\^\= { printf("(%s) - Found BITWISE\_EX\_OR\_ASSIGNMENT\_OPERATOR\n", yytext); }

\|\= { printf("(%s) - Found BITWISE\_IN\_OR\_ASSIGNMENT\_OPERATOR\n", yytext); }

\| { printf("(%s) - Found BITWISE\_OR\_OPERATOR\n", yytext); }

\<\< { printf("(%s) - Found BITWISE\_LEFTSHIFT\_OPERATOR\n", yytext); }

\>\> { printf("(%s) - Found BITWISE\_RIGHTSHIFT\_OPERATOR\n", yytext); }

\~ { printf("(%s) - Found BITWISE\_onesCOMPLEMENT\_OPERATOR\n", yytext); }

\>\>\> { printf("(%s) - Found BITWISE\_RIGHTSHIF\_ZERO\_OPERATOR\n", yytext); }

\-\> { printf("(%s) - Found TUPLE\n", yytext); }

\[ { printf("(%s) - Found SQUARE\_BRACKET\n", yytext); }

\] { printf("(%s) - Found SQUARE\_BRACKET\n", yytext); }

\( { printf("(%s) - Found ROUND\_BRACKET\n", yytext); }

\) { printf("(%s) - Found ROUND\_BRACKET\n", yytext); }

\{ { printf("(%s) - Found BRACE\n", yytext); }

\} { printf("(%s) - Found BRACE\n", yytext); }

\; { printf("(%s) - Found SEMICOLON\n", yytext); }

\, { printf("(%s) - Found COMMA\n", yytext); }

\: { printf("(%s) - Found COLON\n", yytext); }

\. { printf("(%s) - Found DOT\n", yytext); }

\& { printf("(%s) - Found AMPERSAND\n", yytext); }

{DECIMAL\_NUMBER} {

parse\_integer(&int\_number, yytext, 10);

printf("Line %d: found decimal number: %d\n", yylineno, int\_number);

}

{HEX\_NUMBER} {

parse\_integer(&int\_number, yytext, 16);

printf("Line %d: found hex number: %d\n", yylineno, int\_number);

}

{OCT\_NUMBER} {

parse\_integer(&int\_number, yytext, 8);

printf("Line %d: found oct number: %d\n", yylineno, int\_number);

}

{BIN\_NUMBER} {

parse\_integer(&int\_number, yytext, 2);

printf("Line %d: found bin number: %d\n", yylineno, int\_number);

}

{REAL\_NUMBER} {

parse\_real(&real\_number, yytext);

printf("Line %d: found real number: %f\n", yylineno, real\_number);

}

{REAL\_NUMBER\_EXPONENT} {

parse\_real(&real\_number, yytext);

printf("Line %d: found real exponent number: %f\n", yylineno, real\_number);

}

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("(%s) - Found IDENTIFIER\n", yytext); }

[0-9]\*[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("(%s) - Error IDENTIFIER\n", yytext); }

" " {}

\n {}

%%