SPACE [ \t]

NEWLINE [\n]

SINGLELINECOMMENT \/[/]+.\*

MULTILINECOMMENT \/\\*.\*\\*\/

DIGIT [0-9]

LOWERCASE [a-z]

UPPERCASE [A-Z]

LEFTPARANTHESIS \(

RIGHTPARANTHESIS \)

LEFTCURLY \{

RIGHTCURLY \}

LEFTSQUARE \[

RIGHTSQUARE \]

DOT \.

SEMICOLON \;

COLON \:

COMMA \,

ASSIGN =

EQUALS ==

NOTEQUALs !=

AND \&

OR \|

IMPLIES \->

IFANDONLYIF <\->

NEGATION !

CONNECTIVE [{AND}{OR}{IMPLIES}{IFANDONLYIF}]

CONCATENATION \+

IF if

ELSE else

TRUE true

FALSE false

TRUTH\_VALUE [{TRUE}{FALSE}]

WHILE while

STRING \"([^\\\"]|\\\"|\\\n|\\\\)\*\"

ALPHANUMERIC ({LOWERCASE}|{UPPERCASE}|{DIGIT})

CONSTANT\_IDENTIFIER \"{ALPHANUMERIC}\*\"

CLASS\_IDENTIFIER {UPPERCASE}{ALPHANUMERIC}\*

FUNCTION func

CONSTANT const

VOID void

TRUTHVALUE truthvalue

RETURN\_TYPE {VOID}|{TRUTHVALUE}

PROPOSITION [{CONSTANT\_IDENTIFIER}{IDENTIFIER}]

VARIABLE var

INPUT input{COLON}

OUTPUT output{COLON}

RETURN return

IDENTIFIER {LOWERCASE}{ALPHANUMERIC}\*

%%

{RETURN} printf("<RETURN>");

{LEFTPARANTHESIS} printf("<LEDTPARANTHESIS>");

{RIGHTPARANTHESIS} printf("RIGHTPARANTHESIS");

{LEFTCURLY} printf("<LEFTCURLY>");

{RIGHTCURLY} printf("<RIGHTCURLY>");

{INPUT} printf("<INPUT>");

{OUTPUT} printf("<OUTPUT>");

{VARIABLE} printf("<VARIABLE>");

{FUNCTION} printf("<FUNCTION>");

{CONSTANT} printf("<CONSTANT>");

{SINGLELINECOMMENT} printf("<SINGLELINECOMMENT>");

{MULTILINECOMMENT} printf("<MULTILINECOMMENT>");

{CLASS\_IDENTIFIER} printf("<CLASS\_IDENTIFIER>");

{SPACE} printf(" ");

{ASSIGN} printf("<ASSIGN>");

{AND} printf("<AND>");

{OR} printf("<OR>");

{IMPLIES} printf("<IMPLIES>");

{NEGATION} printf("<NEGATION>");

{IFANDONLYIF} printf("<IDANDONLYIF>");

{CONSTANT\_IDENTIFIER} printf("<CONSTANT\_IDENTIFIER>");

{TRUE} printf("<TRUE>");

{FALSE} printf("<FALSE>");

{RETURN\_TYPE} printf("<RETURN\_TYPE>");

{IDENTIFIER} printf("<IDENTIFIER>");

%%

int yywrap(){

return 1;

}

int main(void){

yylex();

return 0;

}