**Regular Expression for Javascript Scanner**

21400646 Lim Chae Eon

* digit 🡪 [0-9]
* letter 🡪 [a-zA-Z]
* literal 🡪 .[^%\<>]
* keyword id 🡪 var | do | while | if | else | switch | case | break | default | function
* keyword tag 🡪 script\_start | script\_end | h1 | /h1 | h2 | /h2 | h3 | /h3 | ul | /ul | ol | /ol | br/
* keyword logical value 🡪 true | false
* keyword function name 🡪 window | parseFloat | document
* logical operator 🡪 == | <= | >= | < | >
* arithmetic operator 🡪 + | - | \* | / | ++ | --
* assignment operator 🡪 = | += | -=
* special\_character 🡪 . | , | ; | : | ( | ) | { | } | [ | ] | “ | “
* identifier 🡪 letter(digit | letter)\*
* WS (white\_space) 🡪 [new\_line | tab]+
* LN (line\_next) 🡪 [line\_feed carriage\_return | line\_feed]
* Assignment 🡪 [UDIR=AR | AER1 | AER2];
* function\_parameter 🡪 ( AR )
* UDIR(user\_defined\_identifier\_regex) 🡪 (WS)\*(identifier)(WS)\*
* AR(argument\_regex)   
   🡪 (WS)\*[identifier | digit | “literal” | (keyword\_function)(function\_parameter\*)](WS)\*
* CR(Comment\_regex)   
   🡪 (WS)\*//[literal]+(LN)\*
* LER(logic\_expression\_regex)   
   🡪 (WS)\*(identifier | digit)(WS)\*(logical\_operator)(WS)\*(identifier | digit)(WS)\*
* AER1(arithmetic\_expression\_regex)   
   🡪 (WS)\*(identifier)(WS)\*(arithmetic\_operator\_one\_operand)(WS)\*
* AER2(arithmetic\_expression\_regex)   
   🡪 (WS)\*(identifier | digit)(WS)\*(arithmetic\_operator\_two\_operand)(WS)\*(identifier | digit)(WS)

DFA for script tag

<(keyword\_tag)>

NFA to assign identifier

var(WS)+UDIR=AR;

var(WS)+UDIR=AR(,(CR)\*(LN)\*AR)\*(,(LN)\*UDIR)\*;

var(WS)+UDIR(,UDIR)\*=AR;

DFA to assign identifier

var(WS)+UDIR(,UDIR)\*=AR(,(CR)\*(LN)\*AR)\*(,(CR)\*(LN)\*UDIR)\*;

DFA to keyword id functions

while(WS)\*( LER ) (WS)\*(Assignment | keyword id | identifier function\_parameter )(WS)\*;

while(WS)\*( LER ) (WS)\*(LN)\*{(WS)\*(LN)\*(WS)\*(Assignment | keyword id | identifier function\_parameter )(WS)\* ; (WS)\*(LN)\*)+}

*(assignment | keyword\_function(function\_parameter\*))*

if(WS)\* ( LER ) (WS)\* ((Assignment | keyword id | identifier function\_parameter )) (WS)\*;

if(WS)\*( LER ) (WS)\*(LN)\*{(WS)\*(LN)\*(WS)\*(Assignment | keyword id | identifier function\_parameter )(WS)\* ; (WS)\*(LN)\*}+}

else(WS)+ ((Assignment | keyword id | identifier function\_parameter ))(WS)\*;

else(WS)\*(LN)\* { (WS)\*(LN)\*(WS)\*(Assignment | keyword id | identifier function\_parameter )(WS)\* ; (WS)\*(LN)\*)+}

for(WS)\*( | LER ; | LER ; | LER ) (WS)\*((Assignment | keyword id | identifier function\_parameter ))(WS)\*;

for(WS)\* ( | LER ; | LER ; | LER ) (WS)\*(LN)\*{(WS)\*(LN)\*(WS)\*(Assignment | keyword id | identifier function\_parameter )(WS)\*; (WS)\*(LN)+(WS)\*)+}

switch(WS)\* ( identifier ) (WS)\*(LN)\* { (WS)\*(LN)\*(WS)\*

((case “literal”: (WS)\* (LN) (WS)\* ((Assignment | keyword id | identifier function\_parameter ) (WS)\* ; (WS)\*(LN)\*)\* | default: (WS)\* (LN) (WS)\* ((Assignment | keyword id | identifier function\_parameter ) (WS)\* ; (WS)\*(LN)\*)\*) }