2 ) EBNF 구문

<translation\_unit> -> { <external\_decl> }+

<external\_decl> -> ( <function\_definition> | <decl> )

<function\_definition> -> [ <decl\_specs> ] <declarator> [ <decl\_list> ] <compound\_stat>

<decl> -> <decl\_specs> [ <init\_declarator\_list> ] ';'

<decl\_list> -> { <decl> }+

<decl\_specs> -> ( <storage\_class\_spec> | <type\_spec> | <type\_qualifier>) [ <decl\_specs> ]

<storage\_class\_spec> -> ( 'auto' | 'register' | 'static' | 'extern' | 'typedef' )

<type\_spec> -> (

'void' | 'char' | 'short' | 'int' | 'long' | 'float'

| 'double' | 'signed' | 'unsigned'

| <struct\_or\_union\_spec>

| <enum\_spec>

| <typedef\_name>

)

<type\_qualifier> -> ( 'const' | 'volatile' )

<struct\_or\_union\_spec> -> <struct\_or\_union> (

<id> '{' <struct\_decl\_list> '}'

| '{' <struct\_decl\_list> '}'

| <id>

)

<struct\_or\_union> -> ( 'struct' | 'union' )

<struct\_decl\_list> -> { <struct\_decl> } <struct\_decl>

<init\_declarator\_list> -> <init\_declarator> { ',' <init\_declarator> }

<init\_declarator> -> <declarator> [ '=' <initializer> ]

<struct\_decl> -> <spec\_qualifier\_list> <struct\_declarator\_list> ';'

<spec\_qualifier\_list> -> { ( <type\_spec> | <type\_qualifier> ) }+

<struct\_declarator\_list> -> <struct\_declarator> { ',' <struct\_declarator> }

<struct\_declarator> -> <declarator>

| <declarator> ':' <const\_exp>

| ':' <const\_exp>

???????

<struct\_declarator> -> (

<declarator>

| <declarator> ':' <const\_exp>

| ':' <const\_exp>

)

<struct\_declarator> -> ( <declarator> [ ':' <const\_exp> ] | ':' <const\_exp> )

둘 중에 더 옳은 표현이 무엇일까요?????

<enum\_spec> -> 'enum' ( <id> [ '{' <enumerator\_list> '}' ] | '{' <enumerator\_list> '}' )

<enumerator\_list> -> <enumerator> { ',' <enumerator>}

<enumerator> -> <id> [ '=' <const\_exp> ]

<declarator> -> [ <pointer> ] <direct\_declarator>

<direct\_declarator> -> ( <id> | '(' <declarator> ')' )

{

(

'[' [ <const\_exp> ] ']'

| '(' <param\_type\_list> ')'

| '(' <id\_list> ')'

| '(' ')'

)

}

<pointer> -> '\*' [ <type\_qualifier\_list> ] [ <pointer> ]

<type\_qualifier\_list> -> { <type\_qualifier> }+

<param\_type\_list> -> <param\_list> [ ',' '...' ]

<param\_list> -> <param\_decl> { ',' <param\_decl> }

<param\_decl> -> <decl\_specs> [ ( <declarator> | <abstract\_declarator> ) ]

<id\_list> -> <id> { ',' <id> }

<initializer> -> ( <assignment\_exp> | '{' <initializer\_list> [ ',' ] '}' )

<initializer\_list> -> <initializer> { ',' <initializer> }

<type\_name> -> <spec\_qualifier\_list> [ <abstract\_declarator> ]

<abstract\_declarator> -> (

<pointer> [<direct\_abstract\_declarator>]

| <direct\_abstract\_declarator>

)

<direct\_abstract\_declarator>-> '(' <abstract\_declarator> ')'

| <direct\_abstract\_declarator> '[' <const\_exp> ']'

| '[' <const\_exp> ']'

| <direct\_abstract\_declarator> '[' ']'

| '[' ']'

| <direct\_abstract\_declarator> '(' <param\_type\_list> ')'

| '(' <param\_type\_list> ')'

| <direct\_abstract\_declarator> '(' ')'

| '(' ')'

<typedef\_name> -> <id>

<stat> -> (

<labeled\_stat>

| <exp\_stat>

| <compound\_stat>

| <selection\_stat>

| <iteration\_stat>

| <jump\_stat>

)

<labeled\_stat> -> (<id> | 'case' <const\_exp> | 'default') ':' <stat>

<exp\_stat> -> [ <exp> ] ';'

<compound\_stat> -> '{' [ <decl\_list> ] [ <stat\_list> ] '}'

<stat\_list> -> { <stat> }+

<selection\_stat> -> ( 'if' '(' <exp> ')' [ <stat> 'else' ] | 'switch' '(' exp ')' )<stat>

<iteration\_stat> -> (

'while' '(' <exp> ')' <stat>

| 'do' <stat> 'while' '(' <exp> ')' ';'

| 'for' '(' [<exp>] ';' [<exp>] ';' [<exp>] ')' <stat>

)

<jump\_stat> -> (

'goto' <id>

| 'continue'

| 'break'

| 'return' [ <exp> ]

)

<exp> -> <assignment\_exp> { ',' <assignment\_exp> }

<assignment\_exp> -> { <unary\_exp> <assignment\_operator> } <conditional\_exp>

<assignment\_operator> -> ( '=' | '\*=' | '/=' | '%=' | '+=' | '-=' | '<<='

| '>>=' | '&=' | '^=' | '|=' )

<conditional\_exp> -> <logical\_or\_exp> [ '?' <exp> ':' <conditional\_exp> ]

<const\_exp> -> <conditional\_exp>

<logical\_or\_exp> -> <logical\_and\_exp> { '||' <logical\_and\_exp> }

<logical\_and\_exp> -> <inclusive\_or\_exp> { '&&' <inclusive\_or\_exp> }

<inclusive\_or\_exp> -> <exclusive\_or\_exp> { '|' <exclusive\_or\_exp> }

<exclusive\_or\_exp> -> <and\_exp> { '^' <and\_exp> }

<and\_exp> -> <equality\_exp> { '&' <equality\_exp> }

<equality\_exp> -> <relational\_exp> {

(

'==' <relational\_exp>

| '!=' <relational\_exp>

)

}

<relational\_exp> -> <shift\_expression> {

(

'<' <shift\_expression>

| '>' <shift\_expression>

| '<=' <shift\_expression>

| '>=' <shift\_expression>

)

}

<shift\_expression> -> <additive\_exp> {

(

'<<' <additive\_exp>

| '>>' <additive\_exp>

)

}

<additive\_exp> -> <mult\_exp> {

(

'+' <mult\_exp>

| '-' <mult\_exp>

)

}

<mult\_exp > -> <cast\_exp> {

(

'\*' <cast\_exp>

| '/' <cast\_exp>

| '%' <cast\_exp>

)

}

<cast\_exp > -> { '(' <type\_name> ')' } <unary\_exp>

<unary\_exp> -> (

<postfix\_exp>

| '++' <unary\_exp>

| '--' <unary\_exp>

| <unary\_operator> <cast\_exp>

| 'sizeof' <unary\_exp>

| 'sizeof' '(' <type\_name> ')'

)

<unary\_operator> -> ( '&' | '\*' | '+' | '-' | '~' | '!' )

<postfix\_exp> -> <primary\_exp> {

(

'[' <exp> ']'

| '(' [ <argument\_exp\_list> ] ')'

| '.' <id>

| '->' <id>

| '++'

| '--'

)

}

<primary\_exp> -> (

<id>

| <const>

| <string>

| '(' exp ')'

)

<argument\_exp\_list> -> <assignment\_exp> { ',' <assignment\_exp> }

<const> -> (

<int\_const>

| <char\_const>

| <float\_const>

| <enumeration\_const>

)